

RTE-IVB

Session Monitor

User's Student Course Book
Volume I



22999-90220

September 1, 1979
updated August 1, 1980

22999-90220 Session Monitor User Student Workbook

The following pages were updated in this manual Aug.80:

1-28	2-35	3-2	4-2	5-10 thru 14	6-3
-32		-7	-6	-16	-12
-34		-9	-32	-18	-13
-35		-14		-21	-26
		-18 thru -45		-22	

7-6	8-9	9-4	10-12	12-20	Title pg.
-7	-13	-6	-32		Chpt. 17

20-2
-3
-6 thru -20

Total=80 pages

HP 1000 RTE-IVB/SESSION MONITOR USER'S COURSE STUDENT WORKBOOK — VOLUME 1

This volume of the Student Workbook is for use during week 1 of the 2 week HP 1000 RTE-IVB/Session Monitor User's Course.

The schedule below indicates the chapters of the Student Workbook to be used during the week and the corresponding lab exercises. The topics to be discussed in each chapter are summarized in the Table of Contents.

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	
8	INTRODUCTION and WELCOME	Review 3. RTE ORGANIZATION	Review 5. PROGRAM DEVELOPMENT	Review 7. USING RTE's SERVICES PROGRAMMATICALLY	Review 10. USING FILES PROGRAMMATICALLY — FMP CALLS	8
9	1. AN INTRODUCTION TO RTE					9
10			LAB 5	LAB 7	LAB 10a	10
11						11
12	LAB 1	LAB 3				12

1	Review 2. USING YOUR RTE SYSTEM	Review 4. FILE MANAGEMENT SYSTEM	Review 6. PROCEDURE FILES	Review 8. INTERACTING WITH YOUR PROGRAM	Review (MORE FMP CALLS)	1
2						2
3				9. TYPE 6 FILES LAB 8, 9		3
4	LAB 2	LAB 4	LAB 6		LAB 10b	4
5						5

TABLE OF CONTENTS — VOLUME 1

CHAPTER

1. AN INTRODUCTION TO RTE
 - A. RTE
 - B. The File Management System
 - C. Session Monitor
 - D. Booting Up RTE
2. USING YOUR RTE SYSTEM
 - A. Introduction to FMGR
 - B. Introduction to Program Development in RTE
3. RTE ORGANIZATION
 - A. BREAKMODE — Interacting with RTE
 - B. RTE Concepts —
memory management,
program management (states),
I/O structure
 - C. Trouble Shooting
 - D. BREAKMODE vs SYSTEM Commands
4. FILE MANAGEMENT SYSTEM
 - A. File Management System Overview
 - B. Using Disc Cartridges
 - C. Using Files
 - D. Accessing Non-disc Devices
5. PROGRAM DEVELOPMENT
 - A. The Program Development Process
 - B. FTN4 and ASMB
 - C. LOADR
 - D. COMPL/CLOAD

TABLE OF CONTENTS — VOLUME 1

CHAPTER

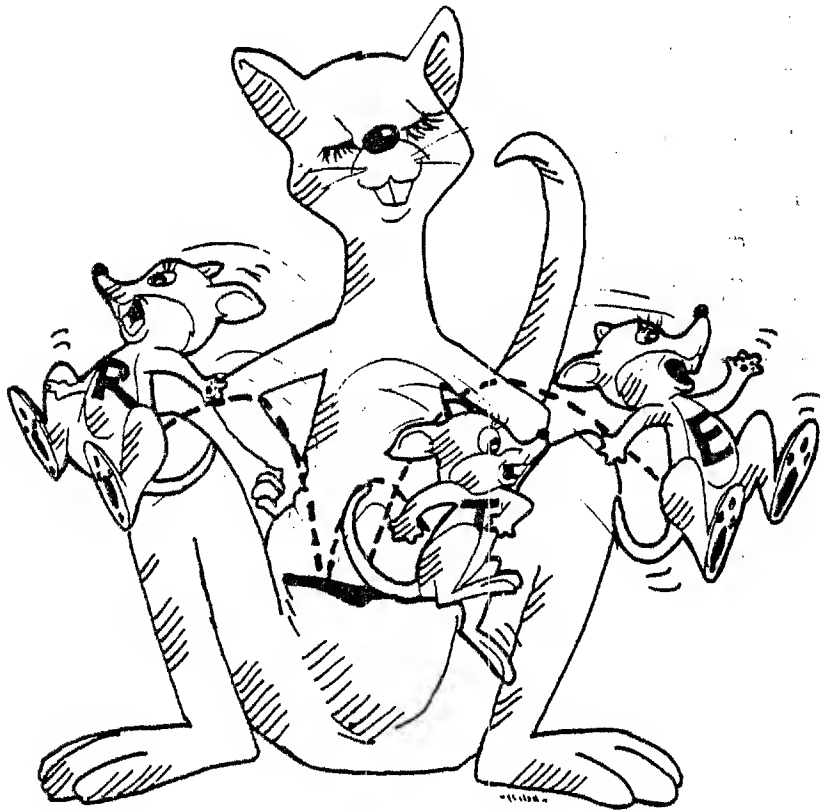
- 6. PROCEDURE FILES**
 - A. What is a Procedure File?**
 - B. Generalized Procedure Files**
 - C. Nested Procedure Files**
 - D. Interacting with Procedure Files**
- 7. USING RTE's SERVICES PROGRAMMATICALLY**
 - A. Introduction to EXEC Calls**
 - B. I/O Processing**
- 8. INTERACTING WITH YOUR PROGRAM**
 - A. Passing Information**
 - B. Suspending Programs**
 - C. Terminating Programs**
- 9. TYPE 6 FILES**
- 10. USING FILES PROGRAMMATICALLY — FMP CALLS**
 - A. Why FMP Calls?**
 - B. How FMP Calls Work**
 - C. Using FMP Calls**
 - D. More on How FMP Calls Work**
 - E. More FMP Calls**

APPENDIX

- A. LAB EXERCISES**
 - Labs 1 to 10**

1

AN INTRODUCTION TO RTE



SECTION

A	RTE	1-3
B	THE FILE MANAGEMENT SYSTEM	1-16
C	SESSION MONITOR	1-24
D	BOOTING UP RTE	1-33

1A. RTE

THE REAL TIME EXECUTIVE OPERATING SYSTEM

AN OPERATING SYSTEM IS:

an organized collection of routines which manages the use of system resources for users and their programs.

RTE MANAGES SYSTEM RESOURCES

SYSTEM RESOURCES INCLUDE:

- **CENTRAL PROCESSING UNIT (CPU)**
 - executes user programs and the routines of the operating system

- **MEMORY**
 - contains the operating system (tables, routines, data areas)
 - contains user programs and their data areas

- **PERIPHERAL DEVICES**
 - are used for secondary storage
 - are used to input or output information to or from the computer

RTE IS A

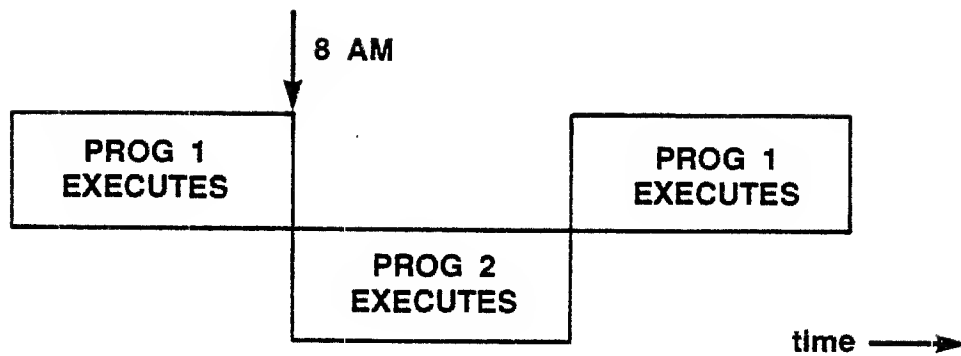
REAL-TIME

MULTIPROGRAMMING

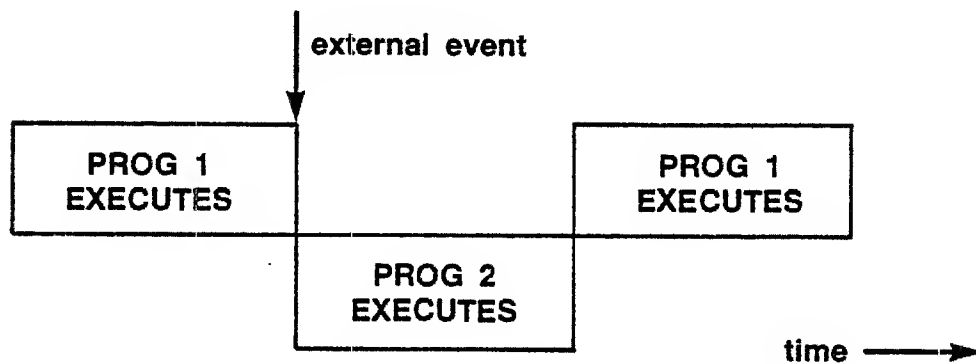
TIME-SLICING

OPERATING SYSTEM

REAL-TIME



- *programs can be scheduled to execute at specific times*



- *programs can execute in response to external events*

RTE IS A REAL-TIME SYSTEM

- RTE maintains a SYSTEM CLOCK which is updated every 10 milliseconds.
- RTE is INTERRUPT DRIVEN.

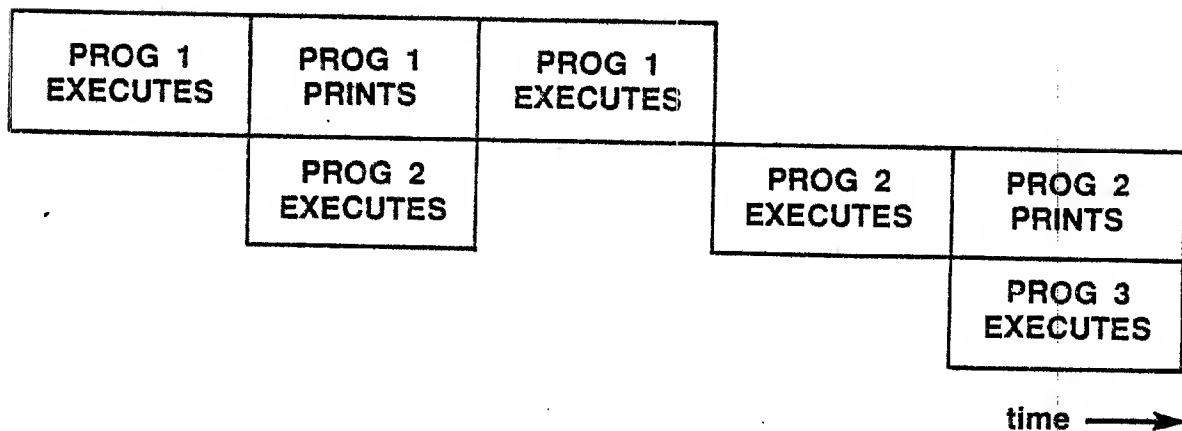
RTE IS INTERRUPT DRIVEN

An INTERRUPT is a voltage pulse that the computer interprets as the signal of an event.

All RTE actions are in response to interrupts:

- 1. You strike a key on a terminal. The terminal sends an interrupt to the computer. RTE responds to the interrupt by printing a command prompt.**
- 2. The line printer finishes outputting a character. It sends an interrupt back to the computer to request the next character.**
- 3. A steam turbine is about to go critical. A temperature sensor sends an interrupt to the computer. RTE recognizes the interrupt and responds by running a program which shuts off the fuel.**

MULTIPROGRAMMING



- while one program is waiting for a data transfer to complete, the CPU can execute another program.
- programs appear to execute in parallel (or concurrently).

RTE IS A MULTIPROGRAMMING SYSTEM

- **PROGRAMS EXECUTE BY PRIORITY**

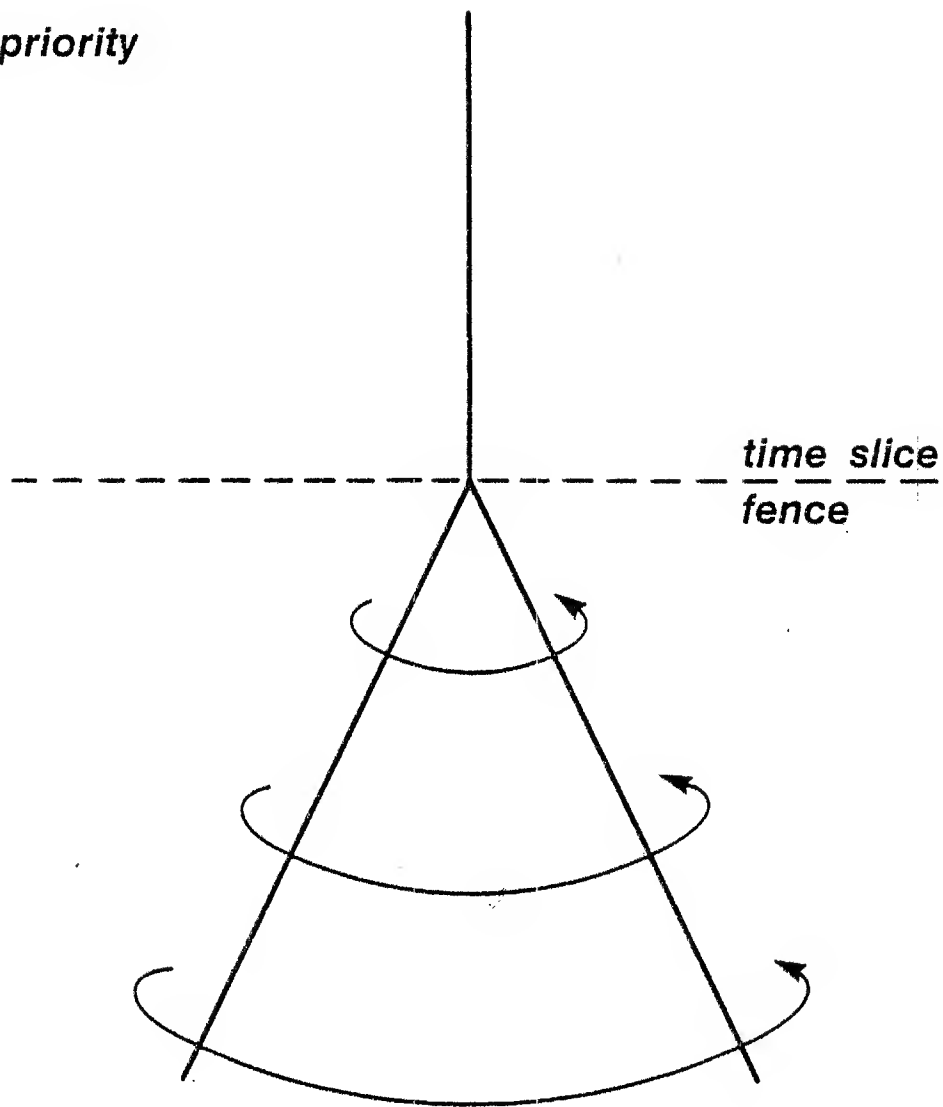
If a program is executing when a higher priority program is scheduled, the higher priority program will begin execution.

- **PROGRAMS ARE SUSPENDED WHILE WAITING FOR DATA TRANSFERS (AMONG OTHER THINGS) TO COMPLETE**

While a program is suspended, RTE will not consider it for execution. When the data transfer completes, RTE will again consider the program for execution, according to its priority.

RTE IS A TIME-SLICING SYSTEM

High priority



Low priority

USING RTE

RTE offers many services for its users, including management of

- programs
- memory
- I/O operations

You can request RTE services with

- **INTERACTIVE** commands entered at your terminal:

*RU, PRDGG

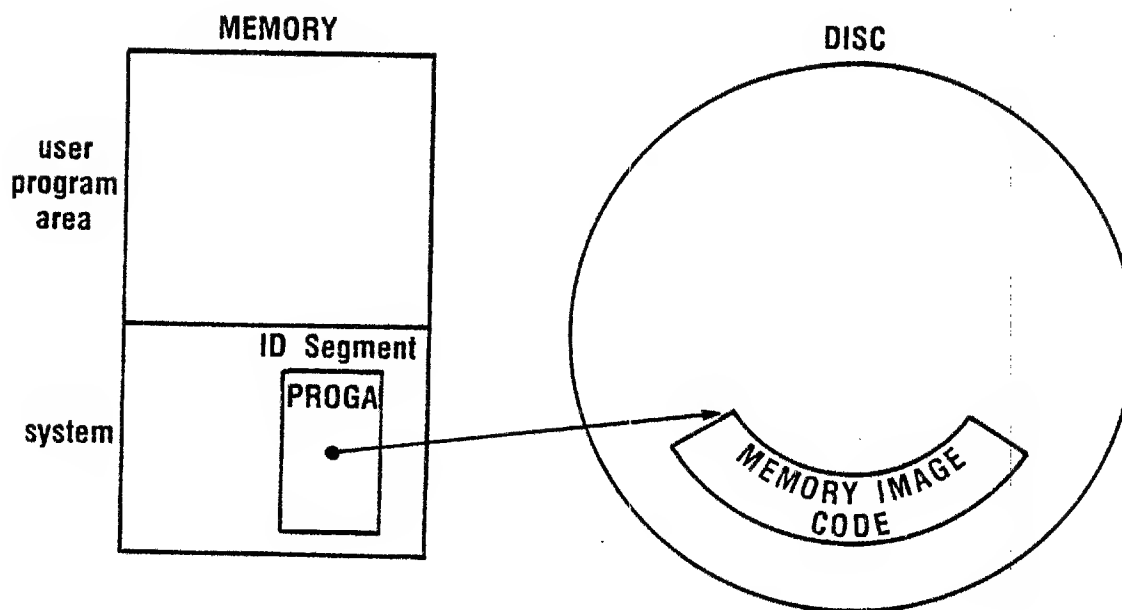
command to name of program
run a program to be run

- **PROGRAMMATIC REQUESTS (EXEC CALLS)** issued by a program:

CALL EXEC (9, NAME)

request code to array containing the
run a program name of program
to be run

RTE RUNS PROGRAMS FOR YOU



In RTE, a program has two parts:

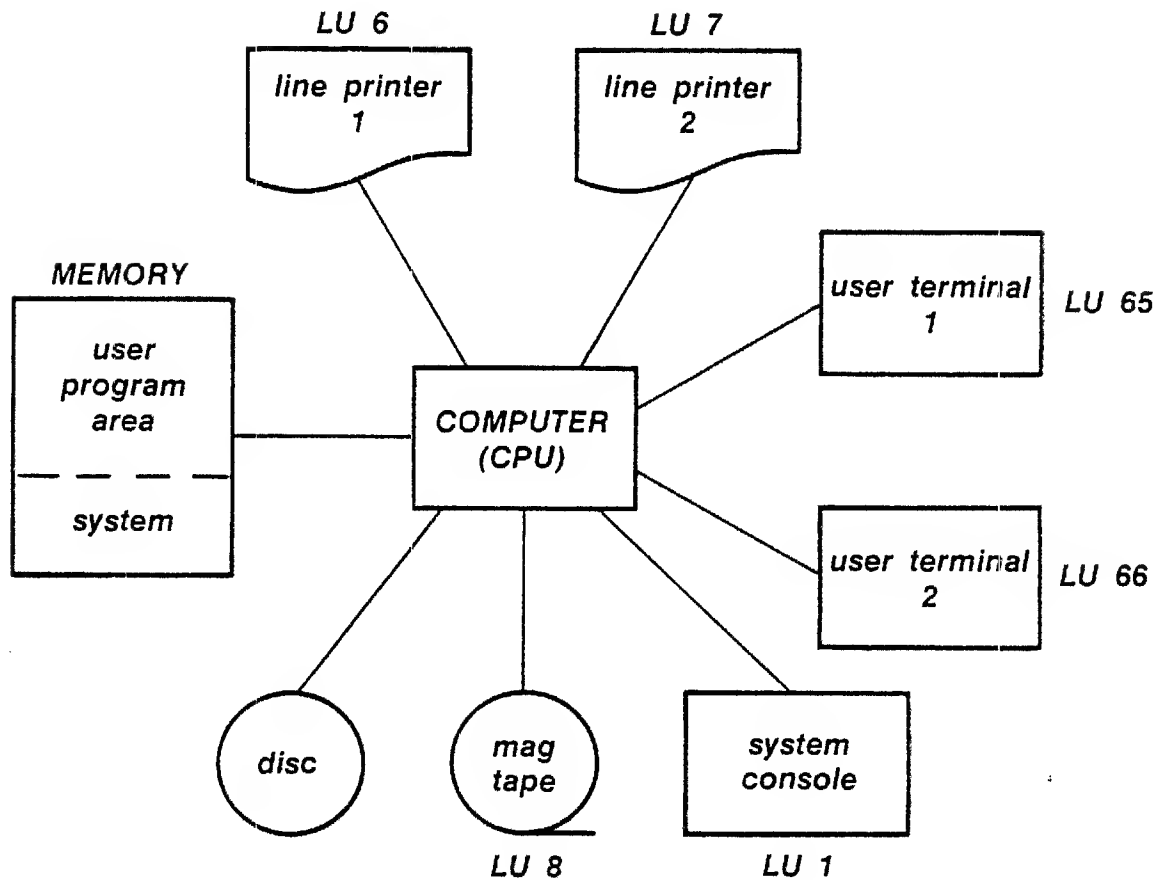
MEMORY IMAGE CODE

- resides on the disc
- contains the program's instructions and data areas
- is created when the program is loaded

ID SEGMENT

- resides in memory
- identifies the program
- contains the location of the program's Memory Image Code on disc
- is filled in (using a blank ID segment) when the program is *Run*

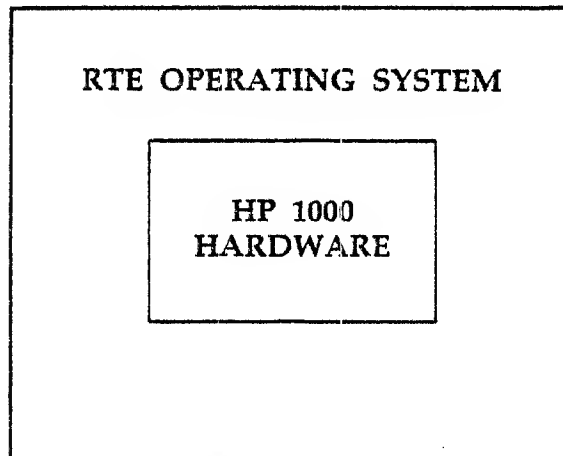
RTE HANDLES I/O FOR YOU



When an RTE system is generated, the System Manager assigns each peripheral device a LOGICAL UNIT (LU) number.

You can then refer to a peripheral device by specifying the appropriate LU.

RTE — A REAL-TIME, MULTIPROGRAMMING, TIME-SLICING OPERATING SYSTEM

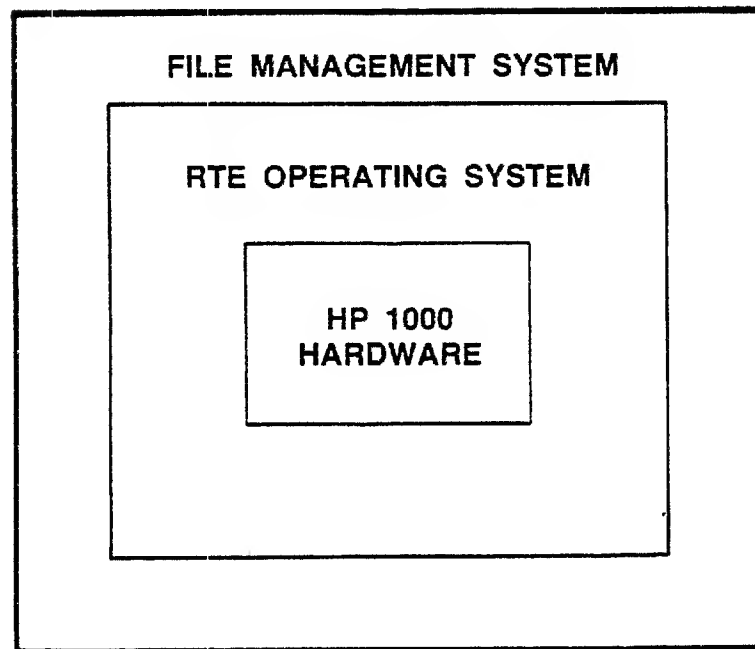


RTE manages system resources

- **program execution**
- **memory management**
- **I/O operations**

via interactive and programmatic commands

1B. THE FILE MANAGEMENT SYSTEM



THE FILE MANAGEMENT SYSTEM

- acts as a user interface to RTE via interactive commands
- manages files for users via interactive and programmatic commands



THE FILE MANAGEMENT SYSTEM ---

FMGR

The program FMGR accepts new commands which

- *interface the user to RTE*
- *allow the user to manipulate files interactively*

You normally use the system through FMGR.

FMP LIBRARY

A set of routines which manage files.

Your programs can manipulate files via calls to the routines in the FMP library.

FMGR uses these routines to do its job.

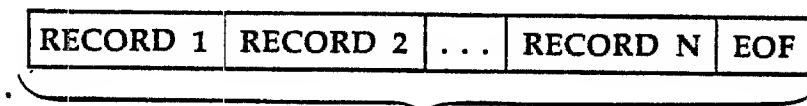
FILES & RECORDS

A FILE is a collection of related pieces of information

- temperature measurements taken last month
- names and addresses of all students in this class
- FORTRAN statements in a FORTRAN source program

A RECORD is an individual piece of information in a file

- a single temperature measurement
- the name and address of one student
- a single statement in a FORTRAN source program

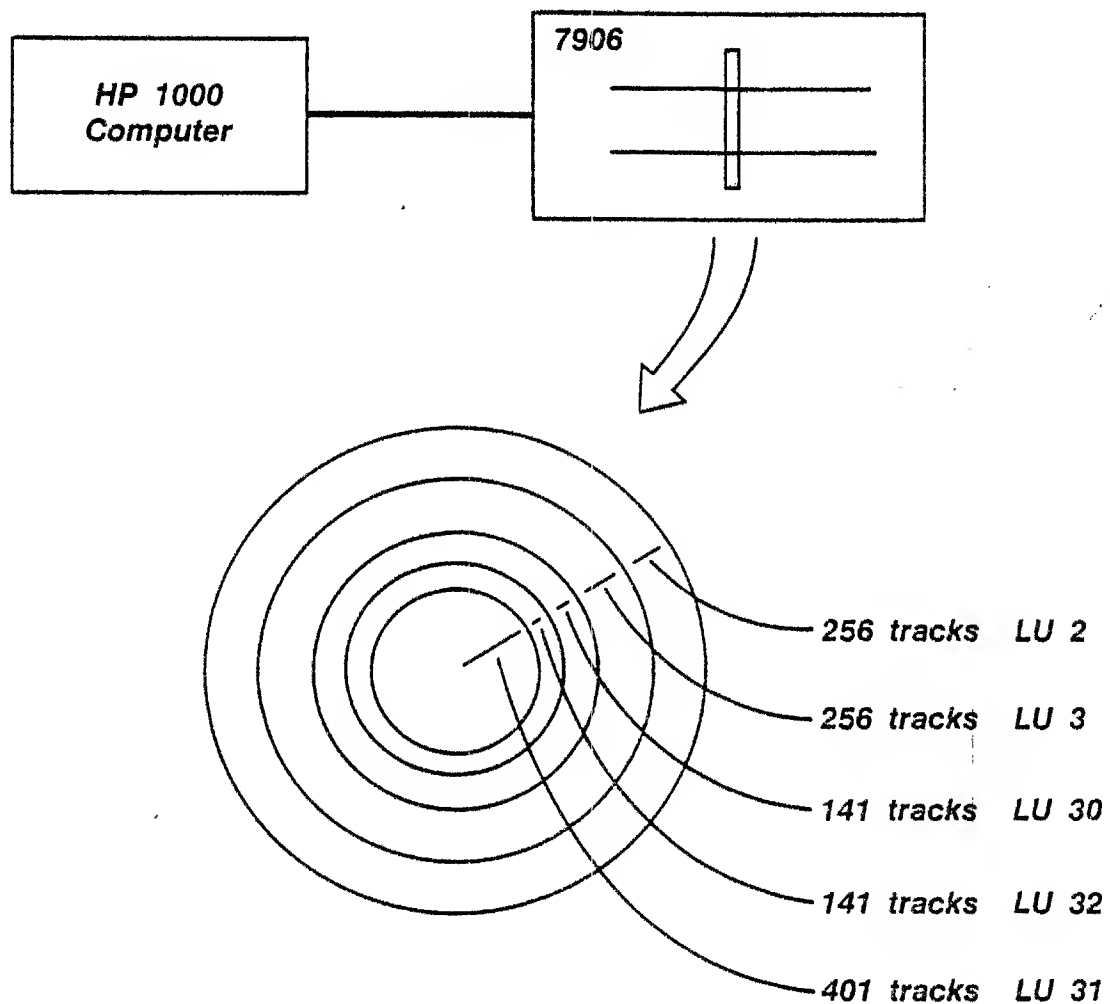


A FILE which might reside

- on disc
- on mag tape
- on cards

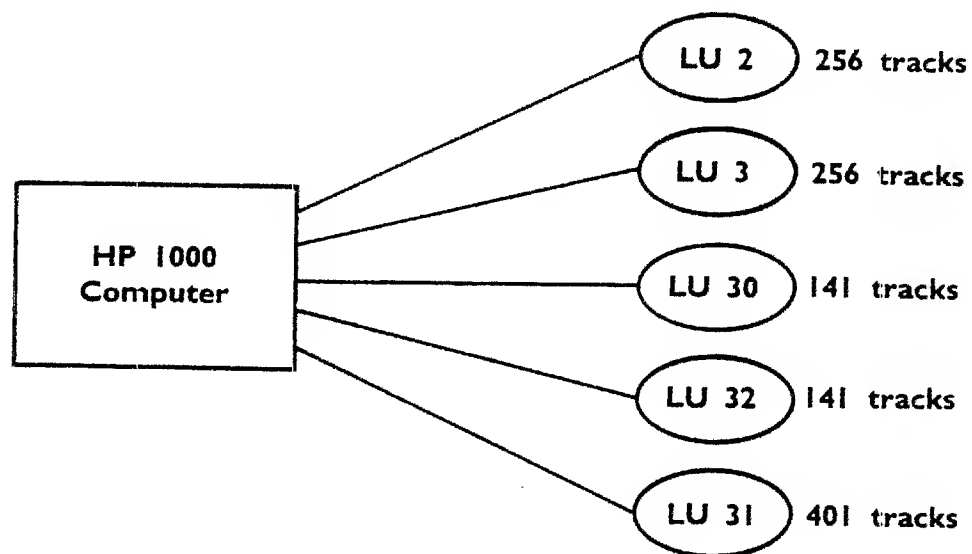
DISC ORGANIZATION

When generating the RTE system, the System Manager divides the disc into several areas. Each of these areas is assigned a Logical Unit number. For example,



(LOGICAL) DISC CARTRIDGES

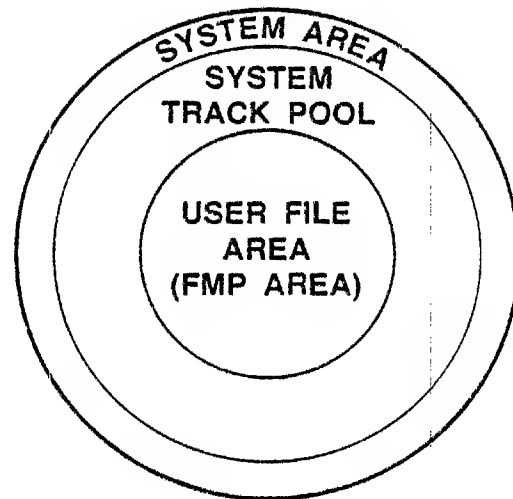
Each disc LU operates independently of the others and can be thought of as a separate "logical" disc.



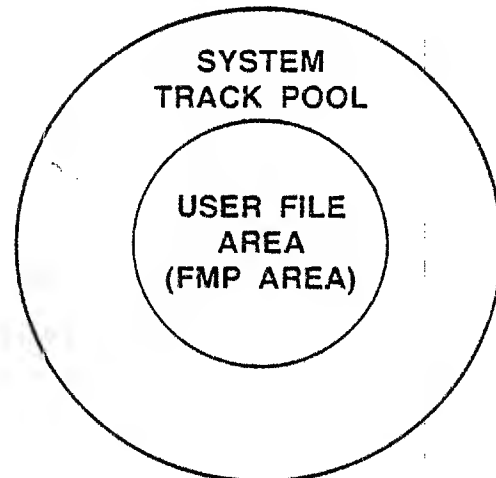
Disc LU's are frequently called DISC CARTRIDGES or CARTRIDGES.

CARTRIDGE ORGANIZATION

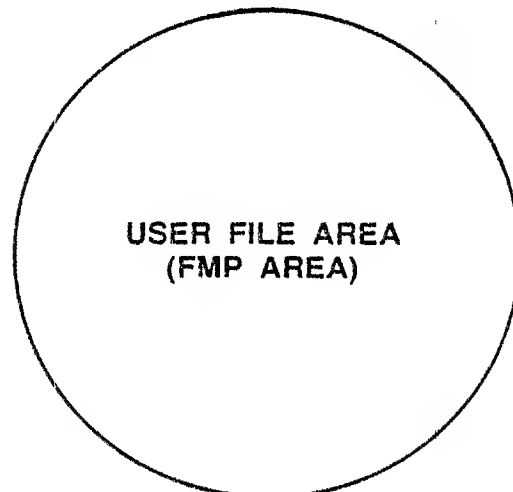
LU 2 —
the system cartridge



LU 3 —
the auxiliary cartridge



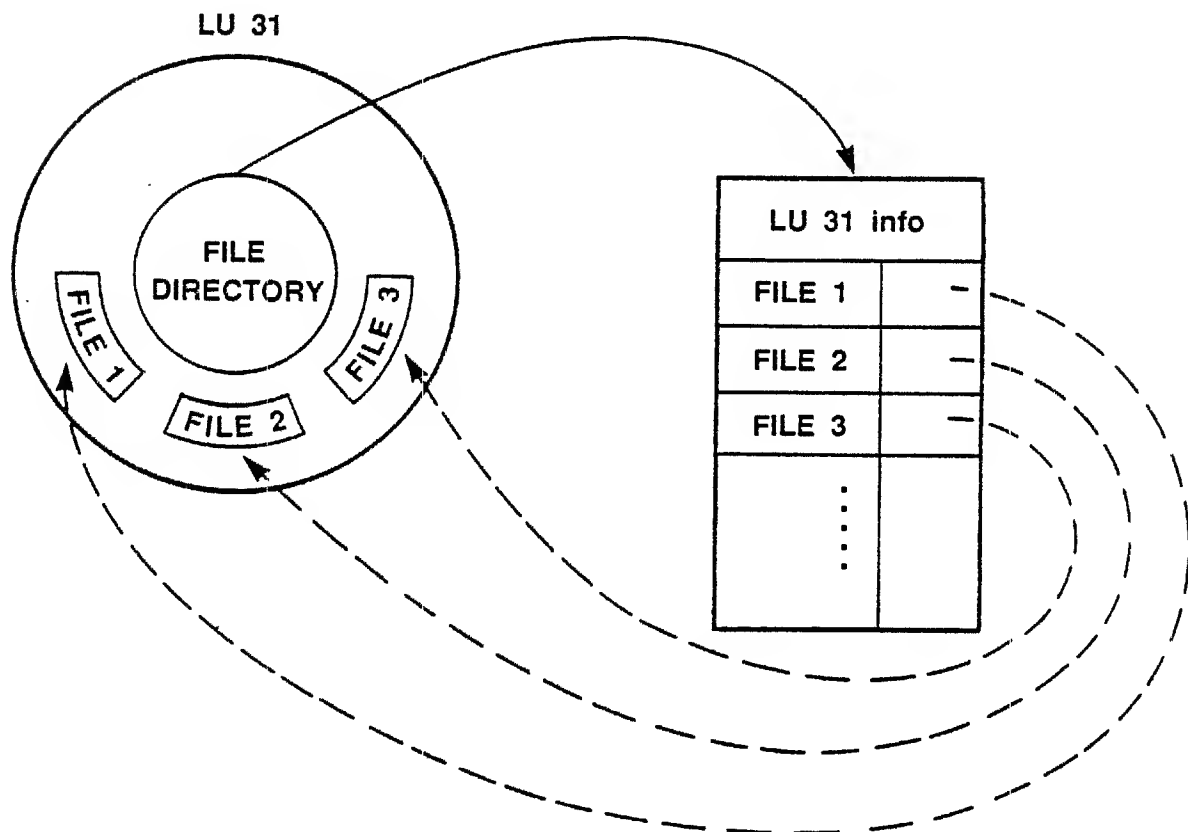
LU x —
a peripheral cartridge



FILE DIRECTORIES

Each cartridge contains a FILE DIRECTORY which

- contains information about that cartridge
- lists the names and locations of all the files residing in the FMP Area of that cartridge



FILES vs PROGRAMS

- **FILE MANAGEMENT SYSTEM**

creates
stores
renames

:
:
:

FILES

- **RTE**

schedules
terminates
suspends

:
:
:

PROGRAMS

- **FILES CONTAIN**

ASCII data
binary data
source code
relocatable code

:
:
:

- **PROGRAMS CONTAIN**
Memory Image Code

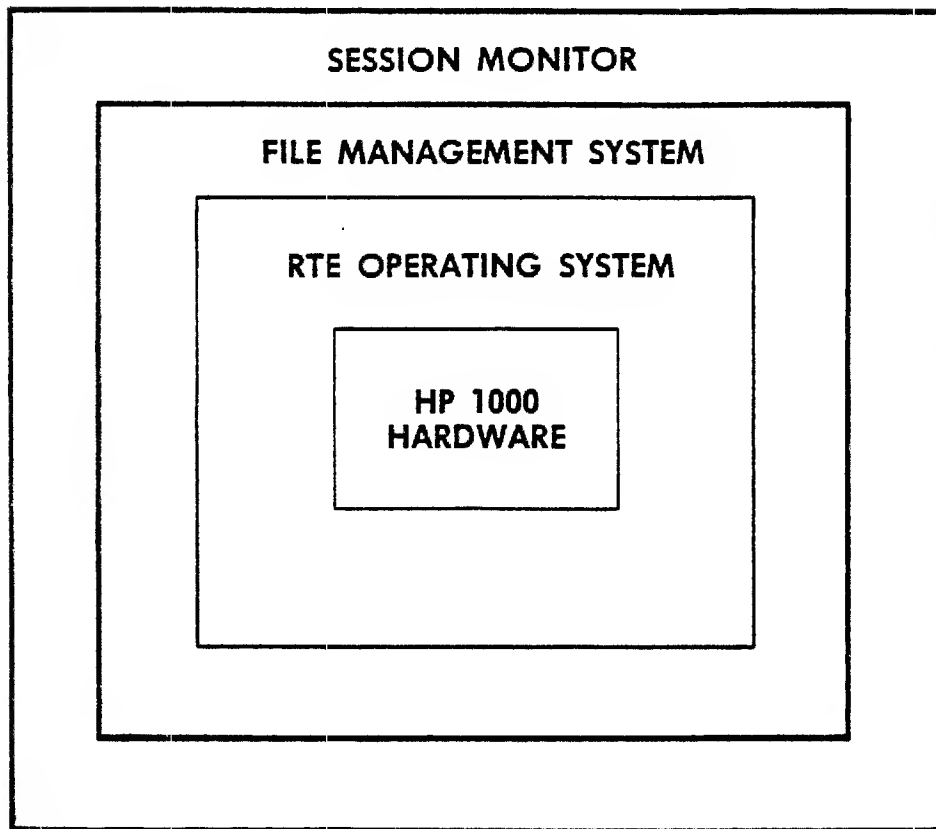
- **FILES RESIDE IN**
the FMP area of a cartridge

- **PROGRAMS RESIDE IN**
the system track pool area of LU 2 or 3

- **FILE DIRECTORIES**
identify FILES

- **ID SEGMENTS**
identify PROGRAMS

1C. SESSION MONITOR



SESSION MONITOR

- restricts access to the system and its resources
- protects users from each other
- provides a friendly multiuser environment

☆ **SESSION MONITOR** ☆

RESTRICTS ACCESS TO THE SYSTEM AND ITS RESOURCES

Each user → must have an account

- may be given restricted access to peripheral devices
- may be given limited use of commands

PROTECTS USERS FROM EACH OTHER

- users can share disc cartridges or have exclusive access to one or more cartridges
- user programs are protected from the activities of other users

PROVIDES A FRIENDLY MULTIUSER ENVIRONMENT

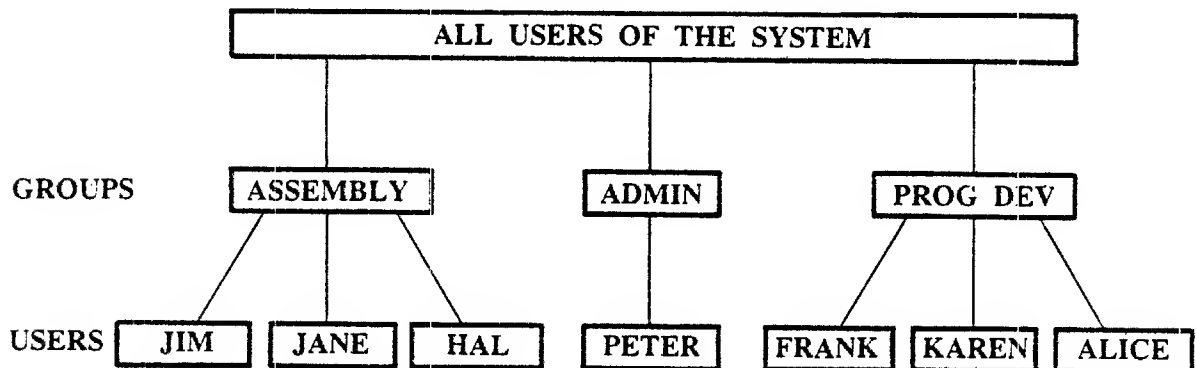
- users do not need to know about "system configuration"
 - every user's terminal is LU 1
- a user at a terminal has the impression of having the system to himself



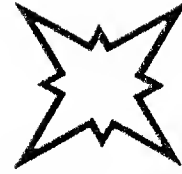
USERS & GROUPS



The System Manager can view the users of an RTE system as both individual users and members of groups of users.



USER ACCOUNTS



- Each user must have an account set up by the System Manager. The user's account describes what the user "can and cannot" do when using the system.
- A user's account consists of a
 - user name — identifies the individual user
 - group name — identifies the user's group
 - password — protects the user's account
 - capability level — restricts the use of commands
 - Session Switch Table (SST) — identifies the peripheral devices which the user can access
- The parameters of each user account are stored in the ACCOUNTS FILE which is maintained by the System Manager.

LOGGING ON

1. Strike a key on a terminal; the system will ask you to log-on.

PLEASE LOG ON: KAREN.PROGDEV
PASSWORD?-----

The system checks the validity of your responses, and then

2. Prints information about the time of log-on and messages from the system manager about the status of the system.

SESSION 65 ON 7:16 PM FRI., 20 JULY, 1979
PREVIOUS TOTAL SESSION TIME:2689 HRS., 48 MIN., 03 SEC.

```
*****  
*                                     *  
*                                     *  
*          WELCOME TO                *  
*      RTE IVB WITH SESSION MONITOR  *  
*                                     *  
*                                     *  
*****
```

THE SYSTEM WILL BE DOWN ON JULY 25 FROM 6 AM TO 9 AM.

3. Lets you know if you have any messages from other users:

MESSAGES WAITING

4. Runs a copy of FMGR at your terminal:

- **FMGR processes a HELLO file set up by the System Manager to greet you.**

HI KAREN, USE THESE LU'S TO REFER TO
I/O DEVICES OR DISC CARTRIDGES:

LU 1 - YOUR TERMINAL
LU 4 - THE LEFT CTU
LU 5 - THE RIGHT CTU
LU 2 - THE SYSTEM CARTRIDGE
LU 3 - THE AUXILIARY CARTRIDGE
LU 6 - THE PRINTER IN ROOM 9
LU 8 - THE MAG TAPE UNIT

- **FMGR then prompts you for a command**

:

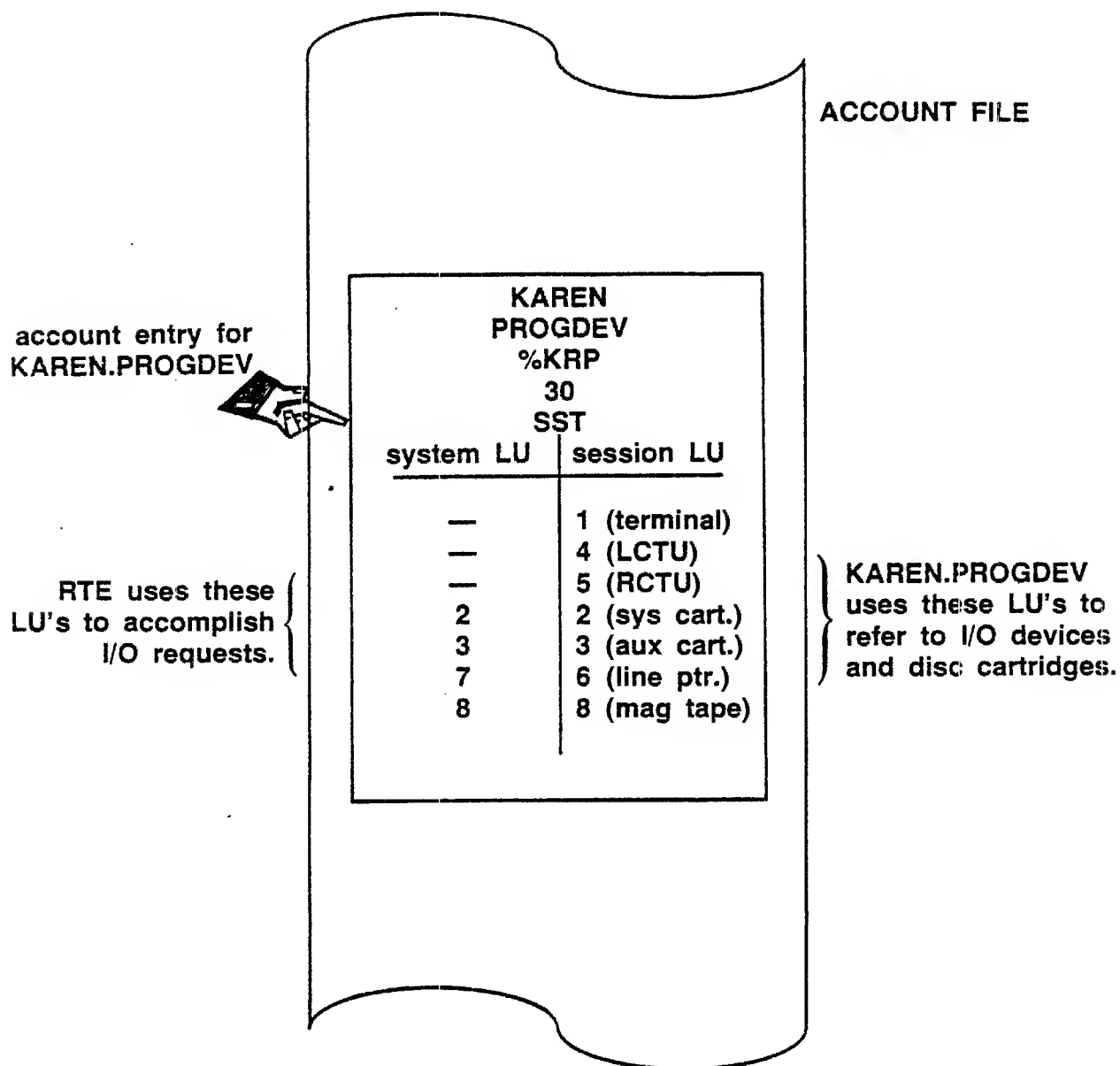


waiting for a command

You are now logged onto a SESSION.

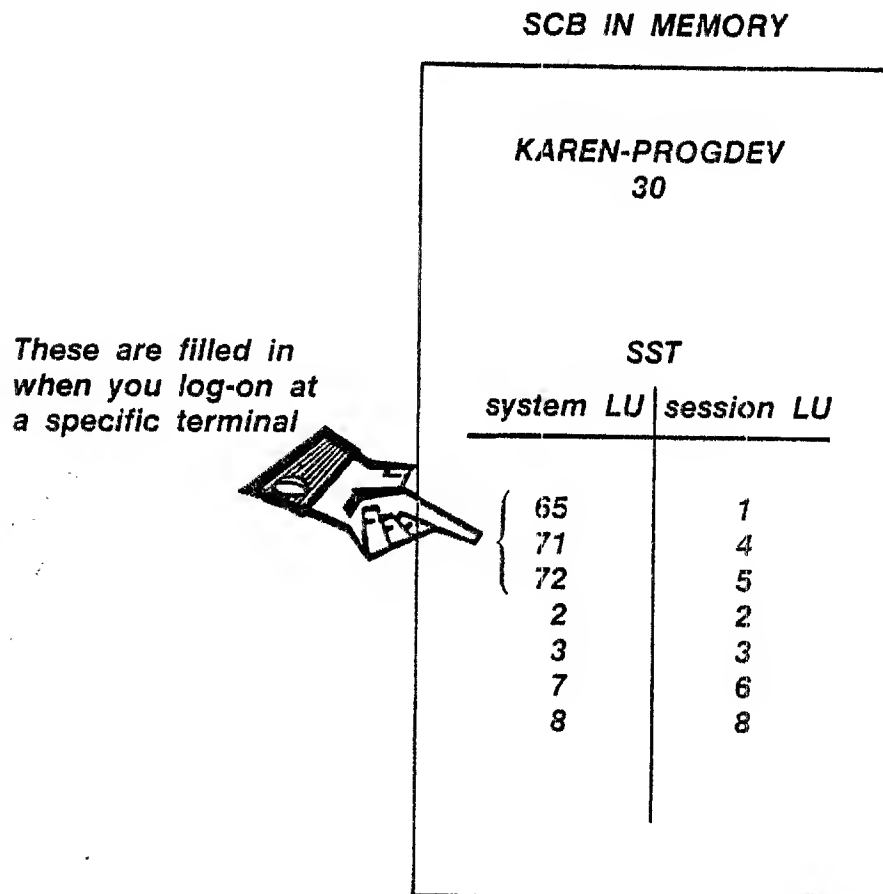
SESSION SWITCH TABLE

When defining a user's account, the System Manager specifies those peripheral devices available to the user. This is done by putting the appropriate LU numbers in the user's Session Switch Table (SST).



SESSION CONTROL BLOCK

If you specify a valid account when logging on, the system creates a Session Control Block (SCB) for you.



The SCB is then used to restrict your access to peripheral devices and interactive commands.

LOGGING OFF

To end your session, use the FMGR EX command to log off.

```
:EX  
$END FMGR  
FMG65 REMOVED
```

```
SESSION    65  OFF   7:23 PM  FRI., 20  JULY, 1979  
CONNECT TIME:                00 HRS.,   07 MIN.,   06 SEC.  
CPU USAGE:                   00 HRS.,   00 MIN.,   01 SEC., 40 MS.  
CUMULATIVE CONNECT TIME:2689 HRS.,   55 MIN.,   09 SEC.
```

```
MESSAGES WAITING  
END OF SESSION
```

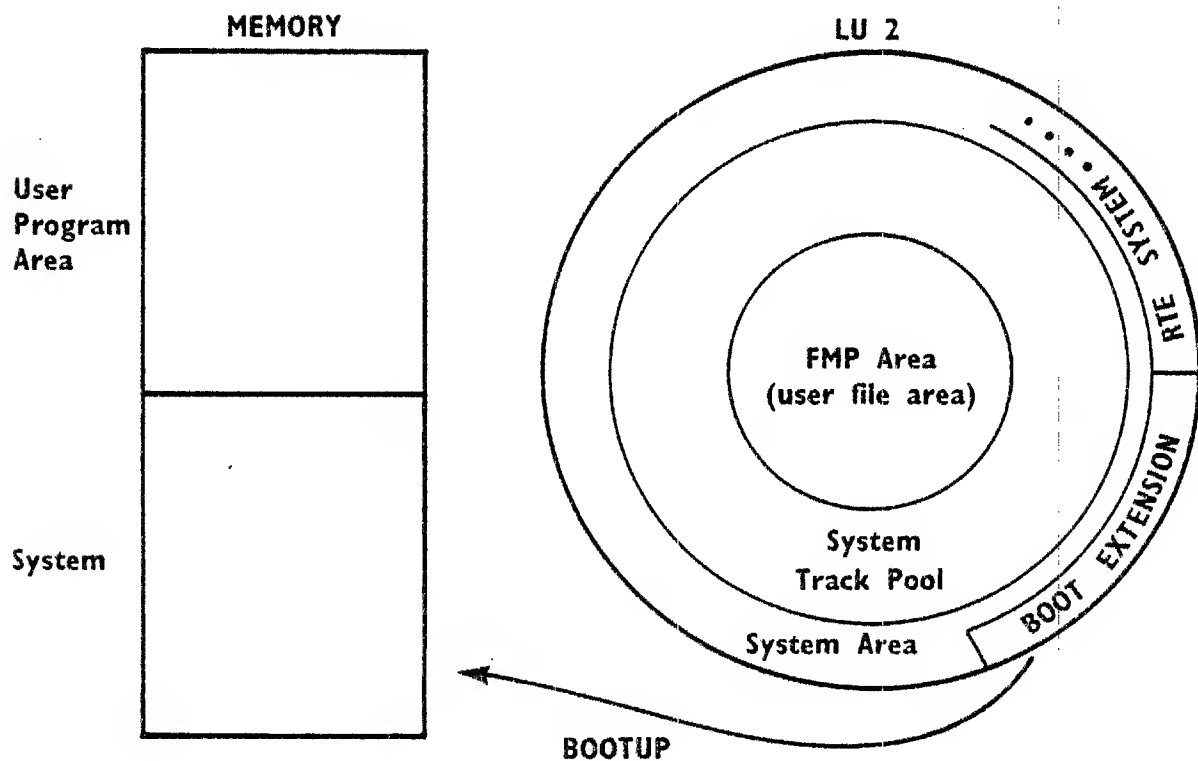
The system logs you off by:

1. terminating any active programs you have.
2. releasing any system resources allocated to you (e.g. releasing any ID segments belonging to your programs; releasing your SCB).
3. posting your connect time, CPU usage time and log off time to your account entry in the ACCOUNTS FILE.
4. printing a log off message at your terminal.

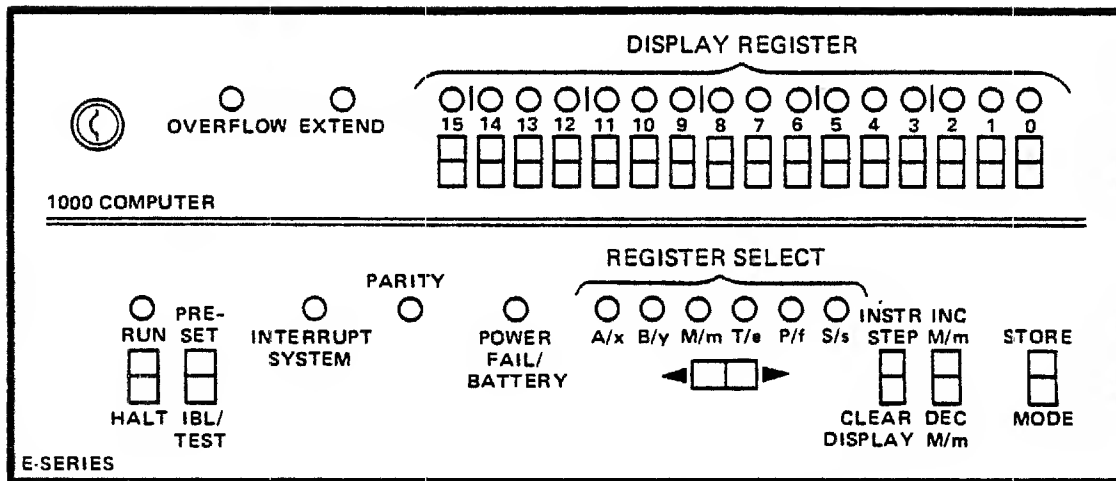
1D. BOOTING UP RTE

RTE resides permanently in the system area of LU 2 (the system cartridge). It must be loaded into memory (booted-up) to execute.

Bootup can be done either manually or automatically.



THE COMPUTER



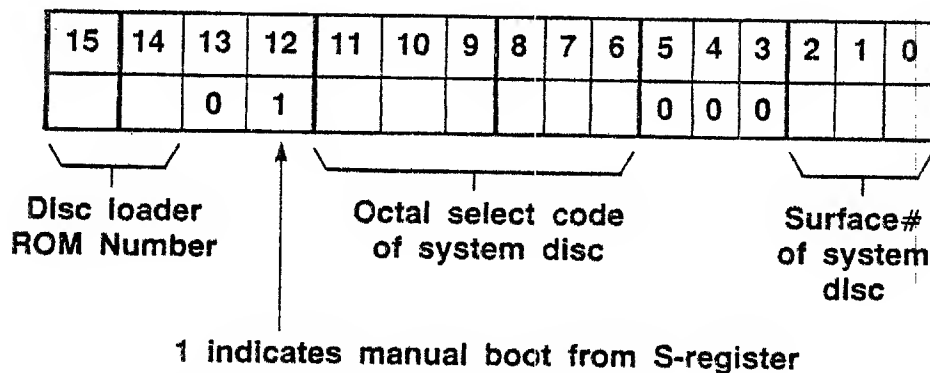
MANUAL BOOTUP is done from the computer's front panel.

MANUAL BOOTUP

With the LOCK/OPERATE switch inside the computer set to OPERATE, the operator must enter information into the S-register to bootup RTE.

Perform these steps to bootup RTE manually:

1. Press HALT
2. Turn on all devices
3. Select the S-Register
4. Enter the following bit pattern into the display register.



5. Press STORE to load the display register into the S-Register
6. Press PRESET
7. Press IBL
8. Press PRESET (again)
9. Press RUN

RTE IS NOW UP AND RUNNING!!!

BOOT INTERNALS

The DISC LOADER ROM contains a program which loads the BOOT EXTENSION from disc into memory.

The BOOT EXTENSION is also a program. It loads the RTE system from disc into memory and causes it to be executed.

The BOOT PROCESS STEPS are:

1. Pressing IBL loads the program in the DISC LOADER ROM into memory.
2. Pressing RUN executes this program causing the BOOT EXTENSION to be loaded into memory. Control is automatically transferred to the BOOT EXTENSION which loads and runs the RTE system.
3. The system displays "SET TIME" on the system console and runs FMGR.
4. FMGR processes a "WELCOM" file set up by the System Manager.

AUTOMATIC BOOTUP



With the LOCK/OPERATE switch inside the computer set to LOCK, RTE will bootup automatically when:

1. the RPL switch is set correctly.
2. the system console is on and on-line.
3. the system disc is on and on-line.
4. the CPU power switch is turned from OFF to ON.

The bootup procedure reads the RPL switch settings to obtain the information needed to boot up RTE.



10C. USING FMP CALLS

The FILE MANAGEMENT SYSTEM supports two types of files, which differ only in their maximum sizes.

	Files allocated by —	
	blocks (STANDARD FILES)	128 block multiples (EXTENDED FILES)
max file size	16383 blocks	32767 x 128 blocks
max record size	32767 words	32767 words
max number of records per file	$2^{15}-1$ records	$2^{31}-1$ records

Since FORTRAN integer variables (1 word) have a maximum value of 32767, double word integers are needed to create or access extended files. Therefore

TWO TYPES OF FMP CALLS

STANDARD FMP CALLS — to manipulate standard files (or access extended files in sequential order only)

EXTENDED FMP CALLS — to manipulate extended files (or standard files)

10

USING FILES PROGRAMMATICALLY FMP CALLS

Section	Page
10A Why FMP Calls?	10-5
10B How FMP Calls Work	10-8
10C Using FMP Calls	10-14
10D More on How FMP Calls Work	10-26
10E More FMP Calls	10-34

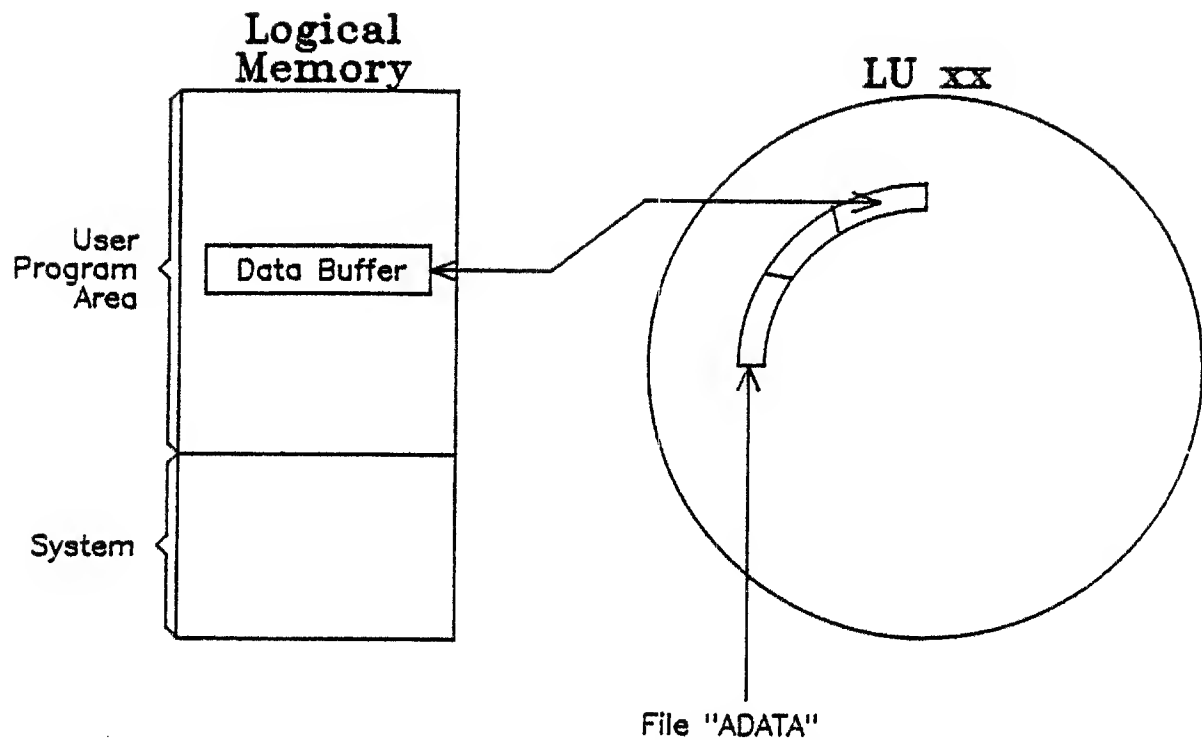
SELF-EVALUATION QUESTIONS

After the end of this module, the student should be able to answer the following self-evaluation questions.

1. List some of the uses of FMP calls.
2. When doing I/O to the disc, how does RTE transfer data?
3. What is a Data Control Block (DCB) and how is it used?
4. What is an extended file?
5. What are the steps to using FMP calls in a program?
6. Explain the difference between update and non-update mode?
7. What is the difference between a logical and a physical read?
8. Why do type 1 files have the fastest transfer rate?
9. What are two ways of accessing a non-disc device as a file?

10A.

WHY FMP CALLS?



FORTRAN READS/WRITES

READ
WRITE (lu,format)-----

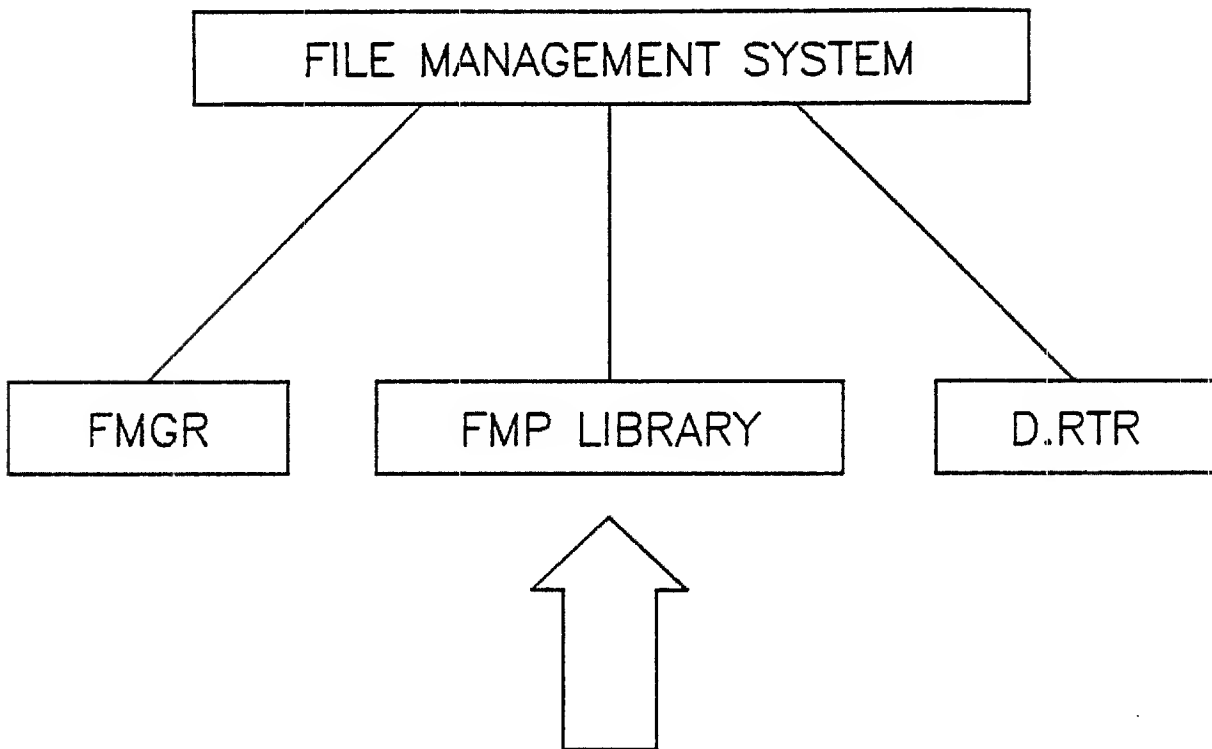
EXEC READS/WRITES

CALL EXEC ($\frac{1}{2}$,lu,-----)

can be a
disc lu

parameters must
specify the exact
track and sector
address of the
desired record.

ACCESSING DISC FILES VIA FMP CALLS



User programs can call these routines
to manipulate disc files.

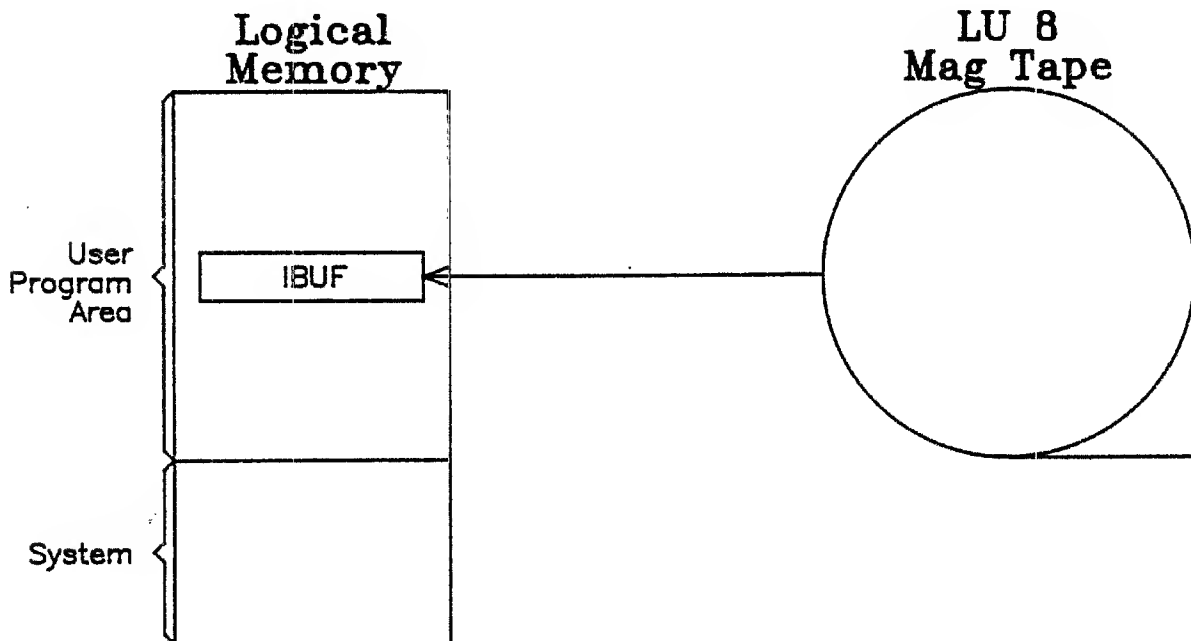
WHAT CAN YOU DO WITH FMP CALLS?

- * Create disc files
- * Access disc files to:
 - read records in sequential or random order
 - write records in sequential or random order
 - position the file at arbitrary locations
- * Purge files
- * Rename files
- * Obtain information about the cartridges in your cartridge list
- * Control non-disc devices by treating them as if they were disc files

10B. HOW FMP CALLS WORK

When a program does I/O to a non-disc device, data is usually transferred directly between the device and a buffer in your program.

For example, consider a file with 100 25-word records on a mag tape.



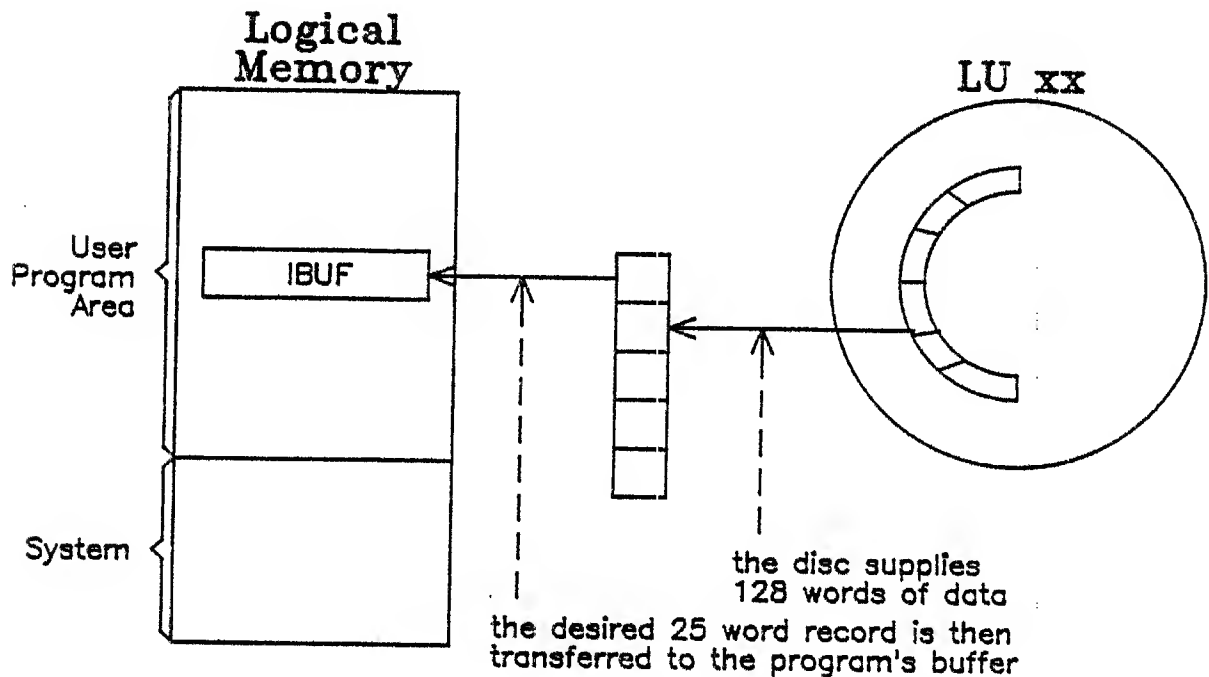
```
ILU = 8  
CALL EXEC (1,ILU,IBUF,25)
```

When you input 25 words of data, the mag tape drive reads 25 words from the tape and stores the data in the program's buffer.

ACCESSING THE DISC

When a program does I/O to the disc, RTE always transfers data in blocks of 128 words, regardless of the size of the records being transferred.

For example, consider a disc file with 100 25-word records:



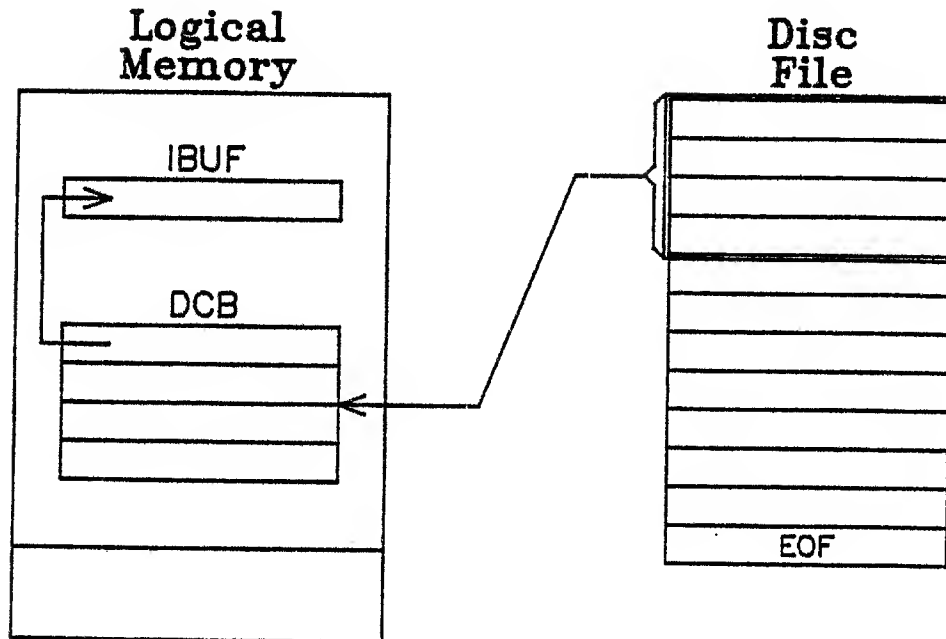
IDEA — why not save all 128 words of data in another buffer in your program? Then, to input the next record, you only need to access memory, not the disc itself.

DATA CONTROL BLOCK

FMP calls do I/O to disc files via a DATA CONTROL BLOCK (DCB) located in your program.

- * The DCB is an intermediate memory buffer used in transferring data between a program and a disc file.
- * Data is temporarily stored in the DCB and transferred to/from the disc only when necessary or explicitly requested.
- * All actions are transparent to the user.

FOR EXAMPLE - READING RECORDS FROM A DISC FILE

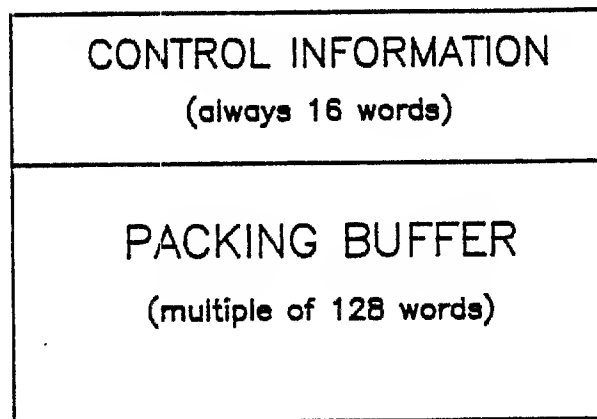


When the program reads the first record of the disc file:

1. a 128 word block of data is input from the disc file into the DCB. This is called a "physical read".
2. the first record is then transferred from the DCB to the program's data buffer. This is called a "logical read".

DCB PARTS

The DCB consists of two parts:



The control information includes:

1. where the file is located on disc,
2. what type of file is being accessed,
3. what records are in the DCB

The packing buffer contains the actual data from the disc file.

ADVANTAGES TO USING FMP CALLS

FMP calls read/write to disc files by name rather than by disc track and sector. Therefore the files can be moved without affecting program execution.

If a piece of data to be read is already in the packing buffer, it is not necessary to do a physical read. This saves time.

FMP ERRORS

Each FMP call requires an error parameter. If an FMP error occurs when the call is executed, the File Management System returns an error code in that parameter. Your programs should always check for errors and respond accordingly.

For example, if a program is writing records to a disc file:

```
      .  
      .  
      .  
      .  
CALL WRITF(IDCIB,IERR,IBUF)  
IF(IERR.LT.0)GO TO 999  
      .  
      .  
      .      (continue on with processing)  
      .  
      .  
C      ERROR REPORTING SECTION  
C  
999  CONTINUE  
      WRITE(1,101)IERR  
101  FORMAT("FMP ERROR",15,"OCCURRED")  
      CALL CLOSE(IDCIB)  
      STOP  
      .  
      .  
      .  
      .
```

STEPS TO USING FMP CALLS

Each program using FMP calls needs the following parts:

1. Declare an array of at least 144 words to serve as the DCB for your FMP calls.
2. Open the disc file to the program by
calling OPEN for an existing disc file or
calling CREAT to create a file and open it.

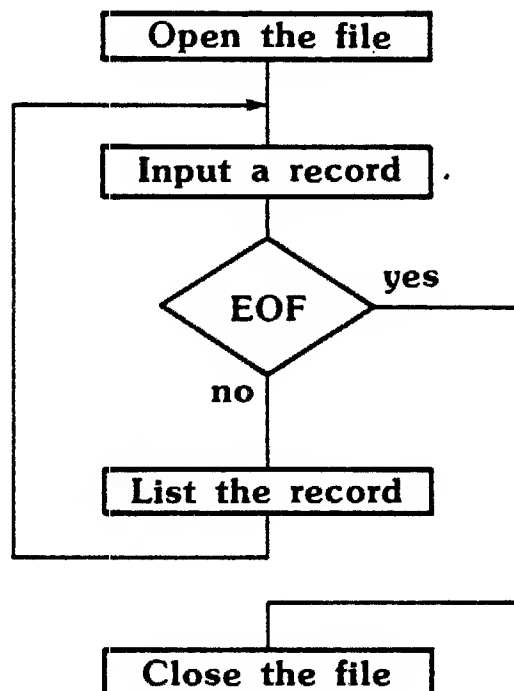
Opening a file sets an open flag in the file directory and associates the user specified DCB with the file.

3. Manipulate the disc file as desired, perhaps by calling READF, WRITEF, etc.
4. Close the file by calling CLOSE to clear the open flag in the file directory and disassociate the DCB from the file.

A PROBLEM TO CONSIDER

Suppose file &PROGA contains a source program, has a security code of "RT" and resides on cartridge LU 47 (CRN SS).

Write a program to list the file at your terminal.



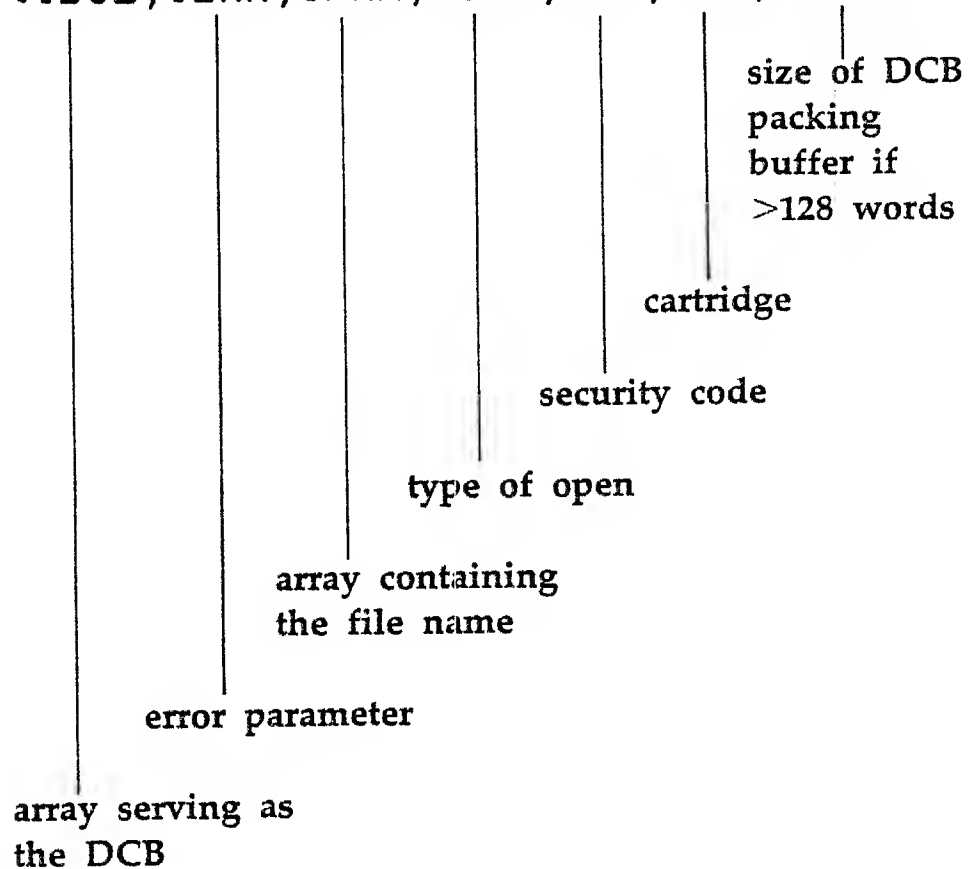
OPENING A FILE

•CALL OPEN•

OPEN will

1. close the DCB if it was left open previously.
2. associate the DCB with the named file and mark the file open.
3. position the file at the first record.

CALL OPEN (IDCB, IERR, INAM, IOPT, ISC, ICR, IDCBZ)



DIFFERENT WAYS TO OPEN A FILE

EXCLUSIVE vs NON-EXCLUSIVE ACCESS

A file opened exclusively may only be accessed by the program that opened the file. A file opened non-exclusively may be shared by up to 7 programs at the same time.

UPDATE vs NON-UPDATE MODE

If a file is opened in update mode, whenever a record is written to the disc file, a physical read is done first to insure that the data in the DCB is current.

If you plan to do random writes to a disc file, you need to open the file in update mode.

If you are only going to read data from or write data to each record in the file sequentially, you only need to open the file in non-update mode.

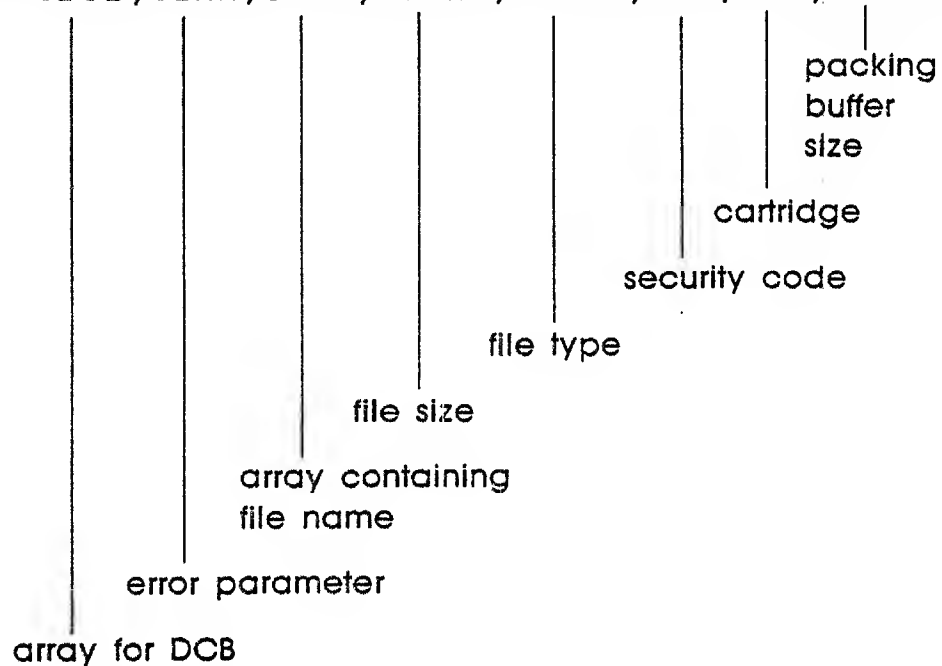
CREATING DISC FILES

•CALL CREAT•
(CALL ECREA)

Creates a file on a disc cartridge by

1. making an entry in the file directory
2. allocating disc space for the file (with no data)

CALL CREAT (IDCB, IERR, INAM, ISIZE, ITYPE, ISC, ICR, IDCZ)



The CREAT (or ECREA) call will

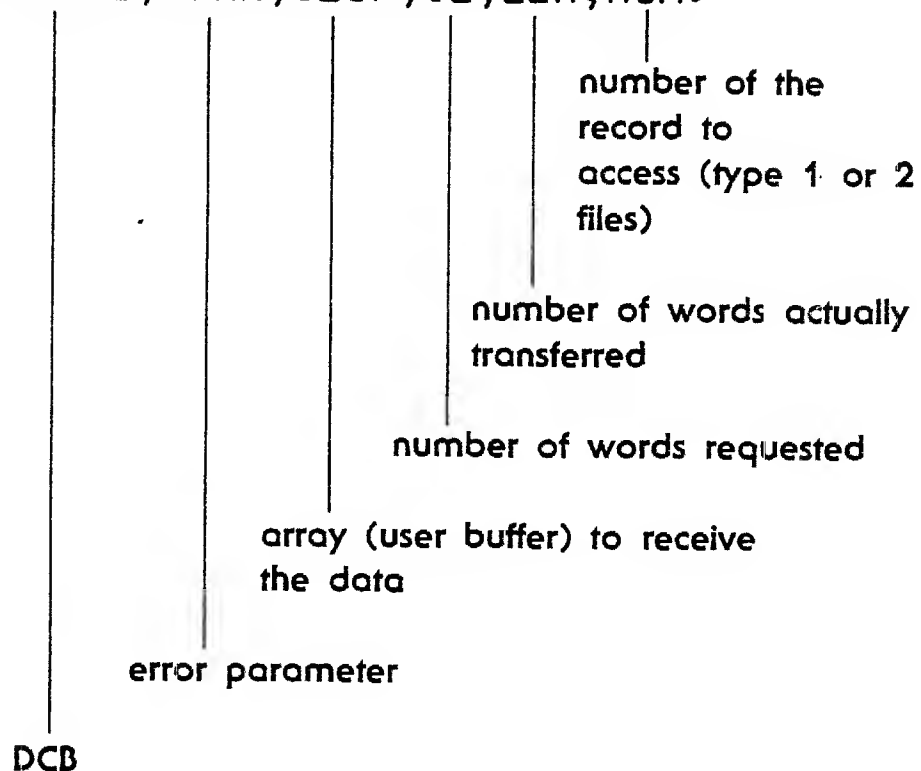
1. close any file currently associated with IDCB
2. create the file
3. open the file exclusively and in update mode. A call to OPEN may be made to change the type of open to non-exclusive or non-update if desired.

READING RECORDS FROM A FILE

•CALL READF•
(CALL EREAD)

Transfers a record from the file currently associated with the DCB to a buffer in the user's program.

CALL READF (IDCB, IERR, IBUF, IL, LEN, NUM)

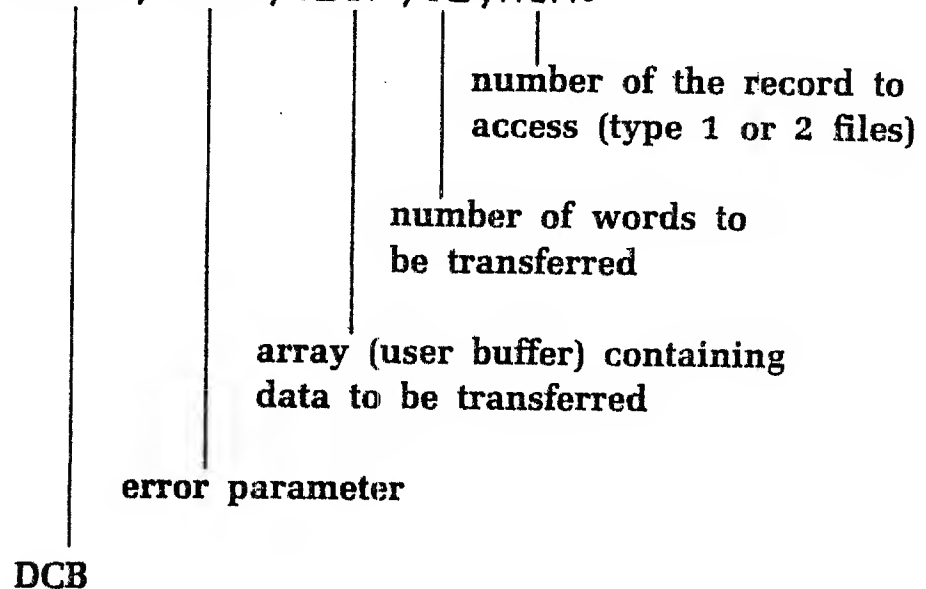


WRITING RECORDS TO A FILE

•CALL WRITF•
(CALL EWRT)

Transfers a record from a buffer in the user's program to the disc file currently associated with the DCB.

CALL WRITF (IDCB, IERR, IBUF, IL, NUM)



CLOSING FILES

•CALL CLOSE•
(CALL ECLOS)

Closes the file associated with the specified DCB by

- writing the DCB to the disc if it was modified
- clearing the open flag in the file directory
- disassociating the DCB from the file

CALL CLOSE (IDCB, IERR, ITRUN)

DCB

allows truncating unused
blocks or extents

error parameter, only needed
if truncating is attempted

Once closed, IDCB can be associated with another file.

A SOLUTION TO OUR PROBLEM

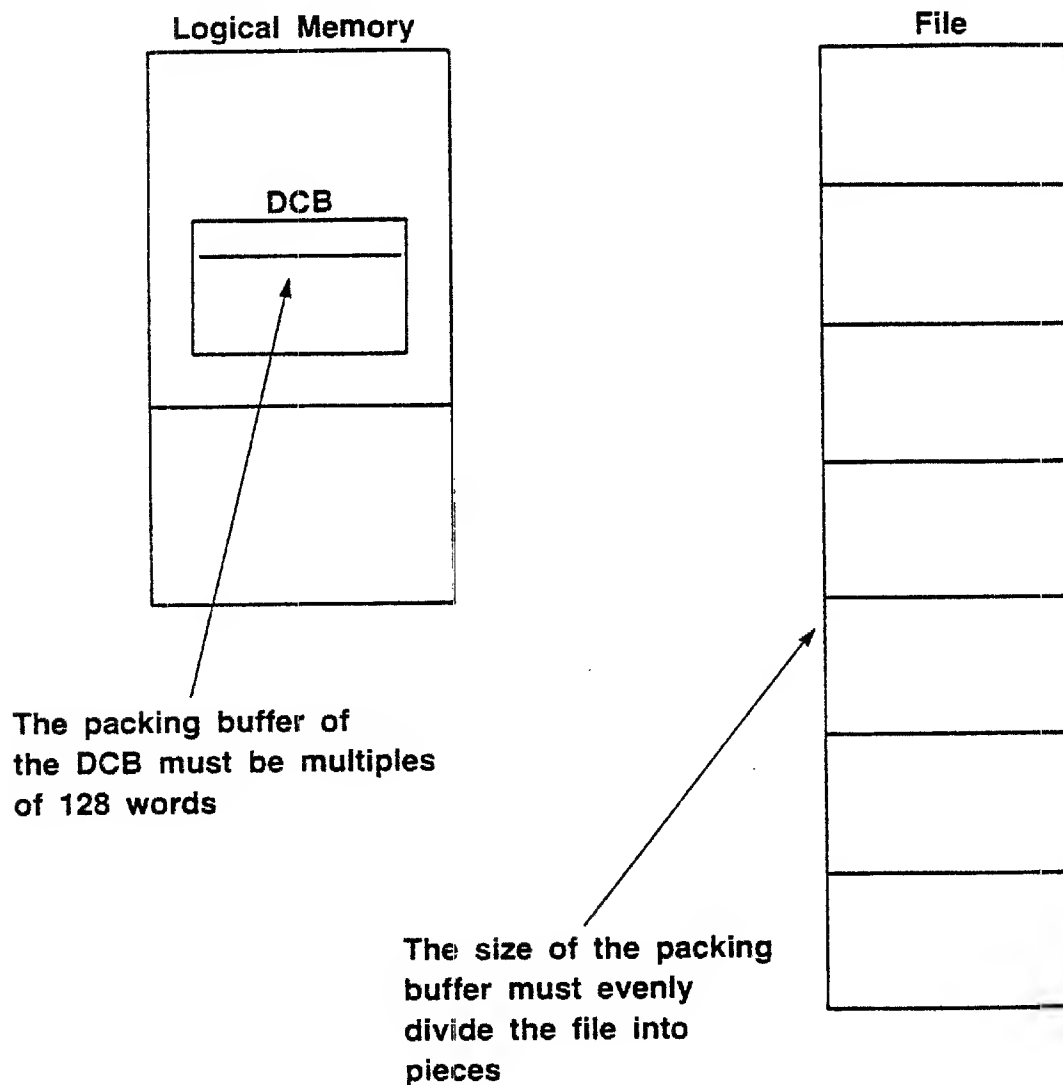
&LISTF T=00004 IS ON CR SS USING 00002 BLKS R=0000

```
0001  FTN4,L
0002      PROGRAM LISTF
0003  C
0004  C      THIS PROGRAM LISTS FILE '&PROGA:RT:SS' ON THE
0005  C      USER'S TERMINAL.
0006  C
0007  C
0008      INTEGER DCB(144), IBUF(40)
0009      INTEGER FILE(3), SEC, CART
0010  C
0011      DATA FILE/2H&P,2HRO,2HGA/, SEC/2HRT/, CART/2HSS/
0012  C
0013  C
0014  C      OPEN THE FILE TO BE LISTED
0015  C
0016      CALL OPEN(DCB,IERR,FILE,0,SEC,CART)
0017      IF(IERR .LT. 0) GOTO 999
0018  C
0019  C
0020  C      READ RECORDS FROM THE FILE,
0021  C      LIST EACH TO TERMINAL UNTIL EOF
0022  C
0023  10    CONTINUE
0024      CALL READF(DCB,IERR,IBUF,40,LEN)
0025      IF(IERR .LT. 0) GOTO 999
0026      IF(LEN .EQ. -1) GOTO 90
0027  C
0028  C
0029  C      LIST RECORD, LOOP BACK FOR NEXT RECORD
0030  C
0031      WRITE(1,101) (IBUF(I),I=1,LEN)
0032  101   FORMAT(1X,40A2)
0033      GOTO 10
0034  C
0035  C
0036  C      AFTER EOF IS FOUND, CLOSE FILE AND QUIT
0037  C
0038  90    CALL CLOSE(DCB)
0039      CALL EXEC(6)
0040  C
0041  C
0042  C      ERROR REPORTING SECTION
0043  C
0044  999   CONTINUE
0045      WRITE(1,102) IERR
0046  102   FORMAT(/"FMP ERROR ",I5)
0047      CALL CLOSE(DCB)
0048      END
0049
```

:

10D. MORE ON HOW FMP CALLS WORK

The DCB acts as a “window” through which your program looks at your file.



ACTUAL vs DECLARED DCB SIZE

The FMP routines will only use part of your declared DCB's packing buffer if the packing buffer does not evenly divide the file.

For example, for a file of 6 blocks (6 x 128 words):

<u>Declared DCB packing buffer size</u>	<u>Actual DCB packing buffer used</u>
128 words	128 words
2 x 128	2 x 128
3 x 128	3 x 128
4 x 128	3 x 128
5 x 128	3 x 128
6 x 128	6 x 128
7 x 128	6 x 128

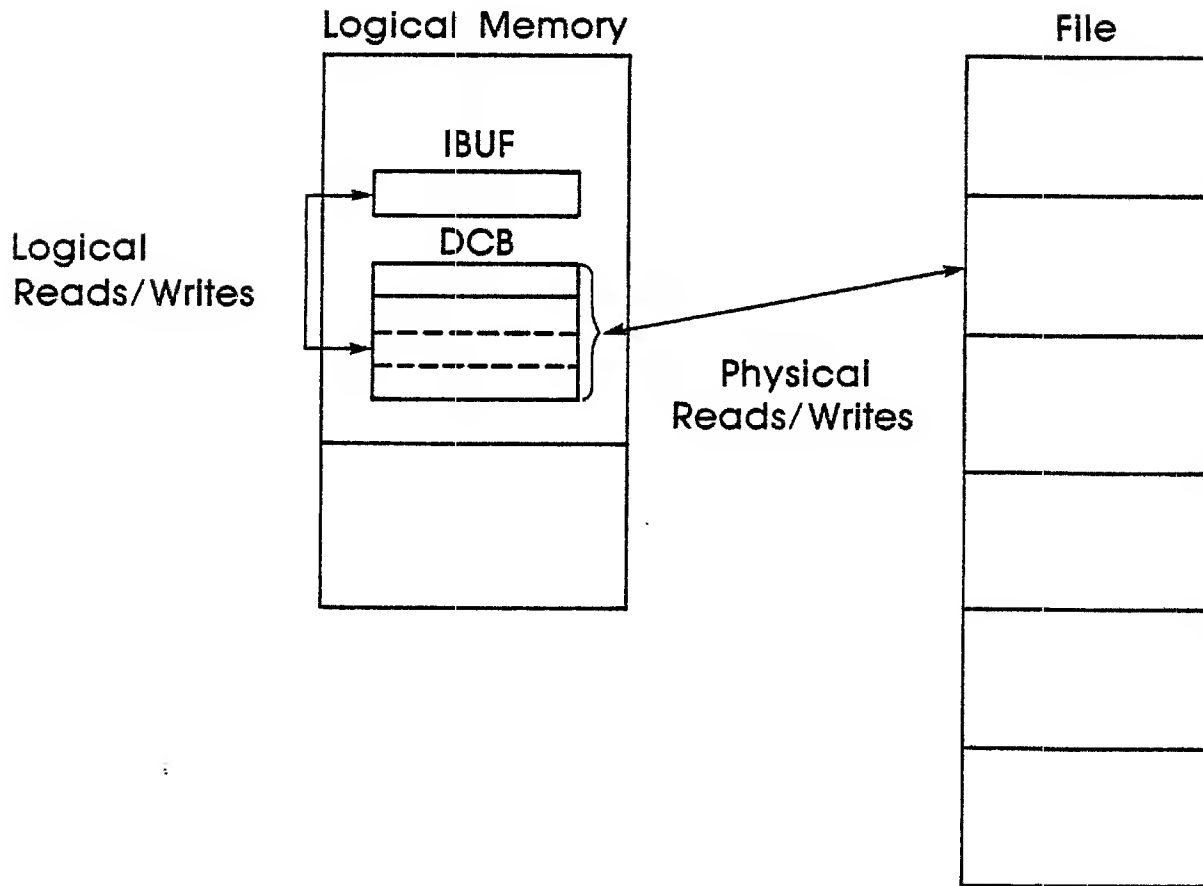
The size of the "actual DCB" used can be obtained by:

$$IUSED = IDCBS (IDCB)$$

Actual DCB size used
(control words + packing buffer)

DCB associated with your file

LOGICAL vs PHYSICAL READS/WRITES



When a program reads or writes a record, a "logical read or write" is done; "physical reads or writes" are done only if the desired record is not in the current DCB.

WHEN IS A PHYSICAL WRITE DONE?

The contents of the DCB are written to the disc file when:

1. the user's program calls CLOSE (or ECLOS) and the DCB contains modified records.
2. the user's program accesses (i.e., reads/writes/positions to) a record not in the current DCB and the DCB contains modified records.
3. the user's program calls POST. POST will write the contents of the DCB to the disc if the DCB contains modified records and set a flag to indicate that there is no data in the DCB.

CALL POST (IDCB, IERR)

DCB for the file

error parameter

WHEN IS A PHYSICAL READ DONE?



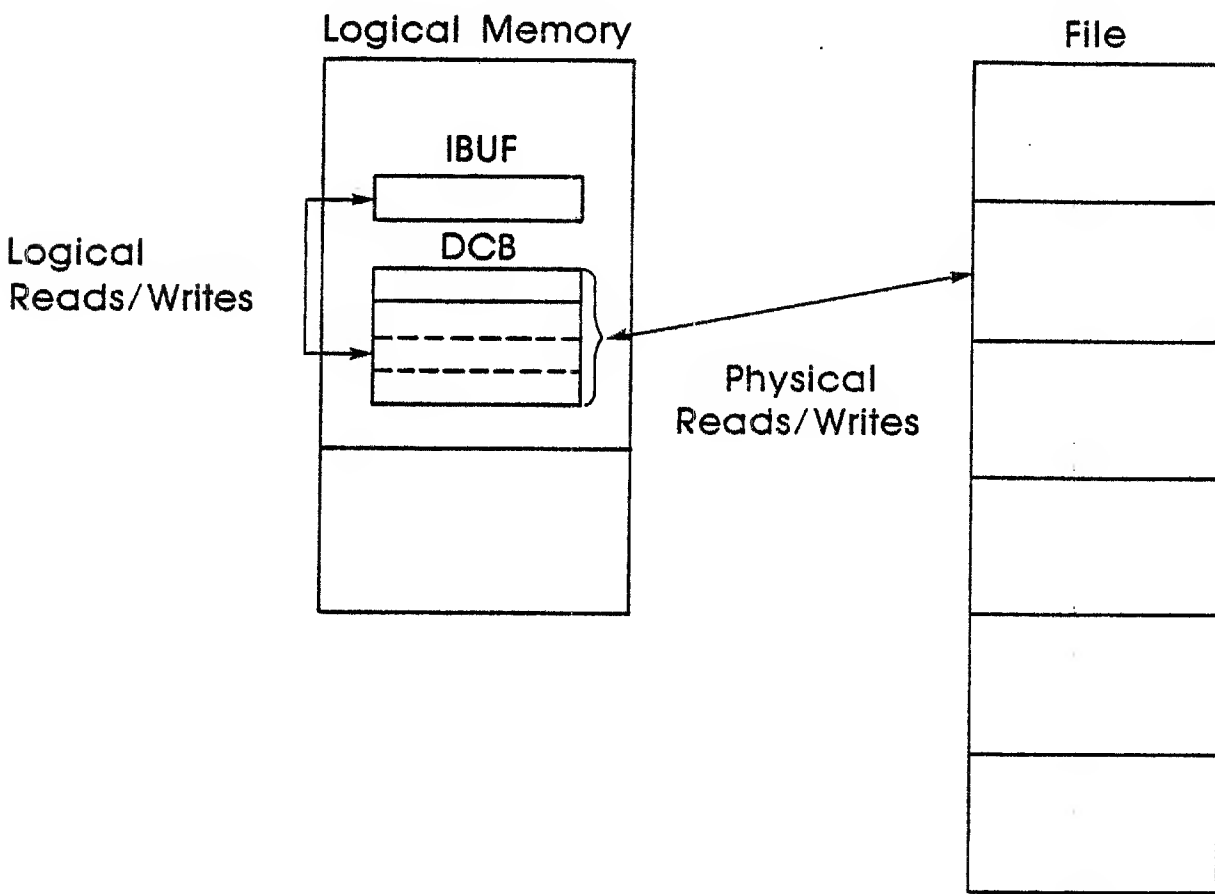
When a DCB is associated with a new “piece” of a disc file, the data may or may not be input from the disc file into the DCB.

- data is not input to the DCB when
 - the file is first opened
 - a program calls WRITF (or EWRIT) and the file was opened in non-update mode
- data is input to the DCB when
 - a program calls READF (or EREAD)
 - a program calls WRITF (or EWRIT) and the file was opened in update mode

TYPE 2,3,... FILE ACCESS

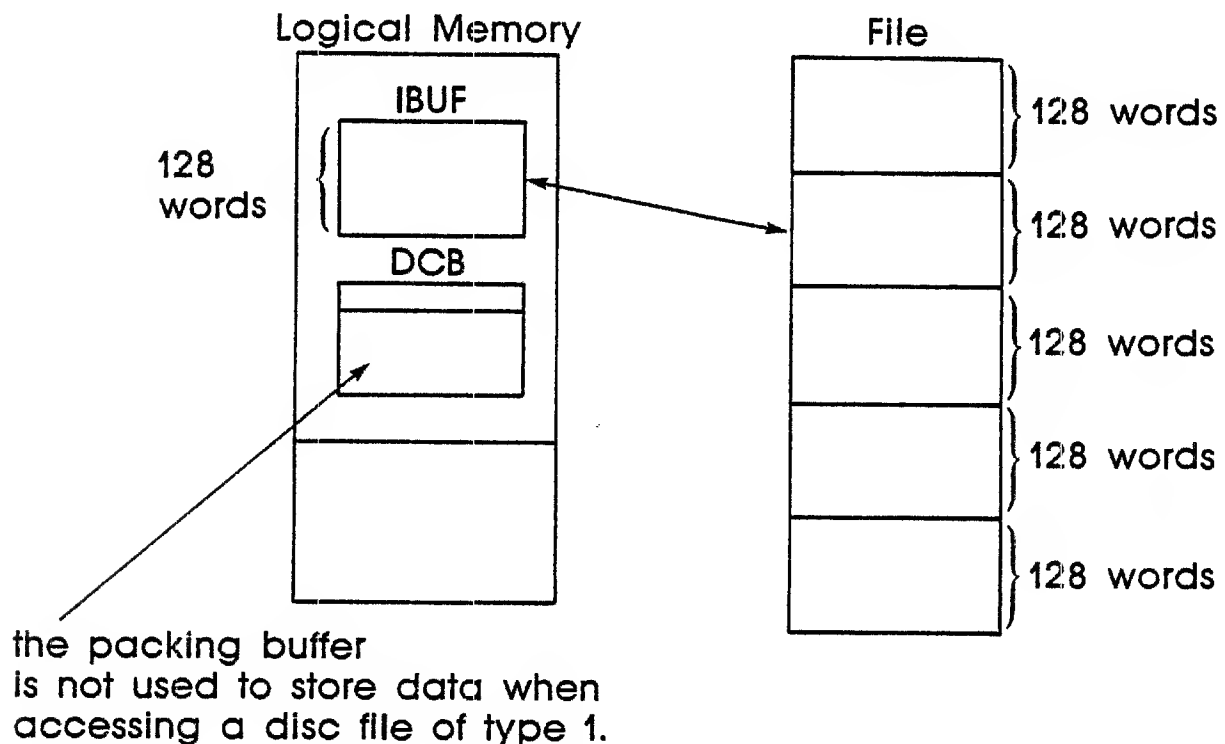
Even though type 2,3,... files have differing characteristics, these types of files are accessed in similar manners:

- records are accessed via the packing buffer in the DCB associated with the file
- data transfers are done record by record



TYPE 1 FILE ACCESS

Each record in a type 1 file is 128 words long. Since the disc drive transfers data in 128 word blocks, the FMP routines transfer data directly between the disc file and the user program's data buffer for type 1 files.

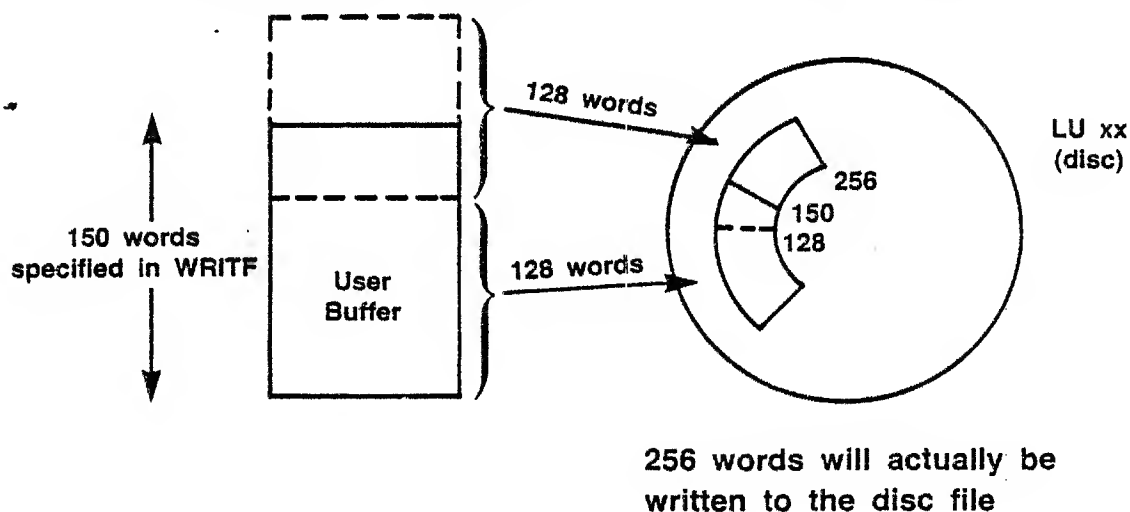


Type 1 files have the fastest transfer rate because transfers are directly to or from the user's buffer.

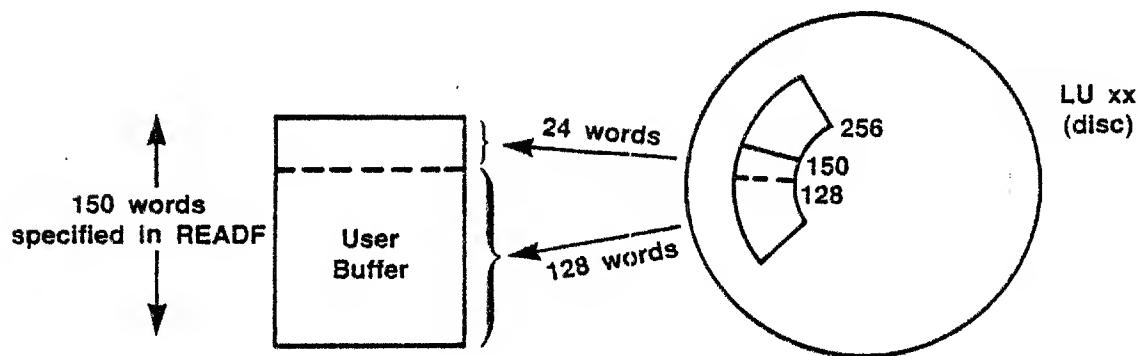
- All files, except type 0 files, may be accessed as type 1 files, i.e., direct transfer through the user's buffer.

- Data transfers with type 1 files are not limited to the 128 word record length.

— for writes, data is transferred in multiples of 128 words



— for reads, data is transferred in the exact amount requested



the disc drive will read 256 words;
the FMP routines will only transfer
150 words to your data buffer.

10E. MORE FMP CALLS

The FMP calls may be grouped as follows:

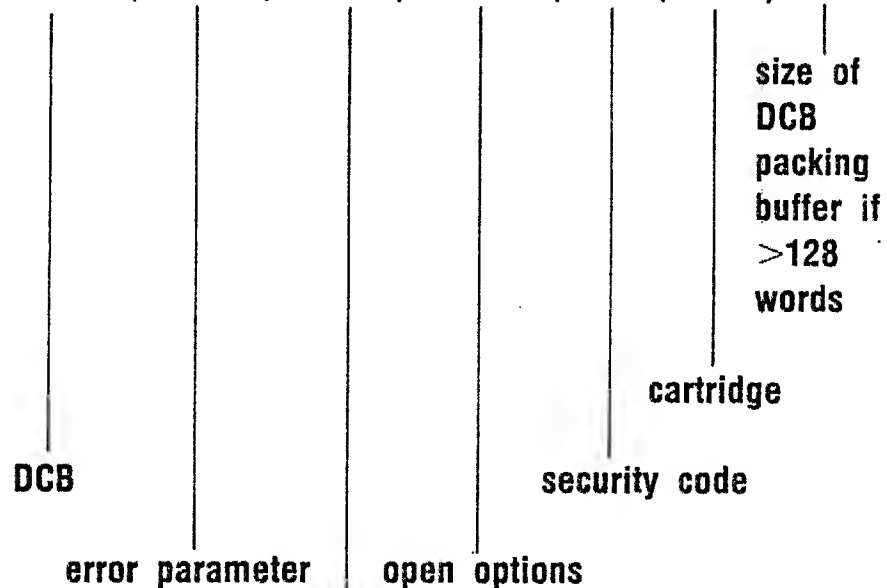
	Standard Call	Extended Call	Special Call
File Definition	CREAT CRETS OPEN CLOSE PURGE	ECREA CRETS ECLOS	OPENF
File Access	READF WRITF	EREAD EWRT	
File Positioning	POSNT RWNDF LOCF APOSN	EPOSN ELOCF EAPOS	
Special Purpose	POST IDCBS NAMEF FSTAT FCONT		

NON-DISC DEVICES

With FMP Calls, you can access non-disc devices just as you would access disc files. This might be done in two ways.

- Via a type Ø file
- Via an OPENF call

CALL OPENF (IDCB, IERR, INAM, IOPTN, ISC, ICR, IDCBZ)



three word array containing a file name

or

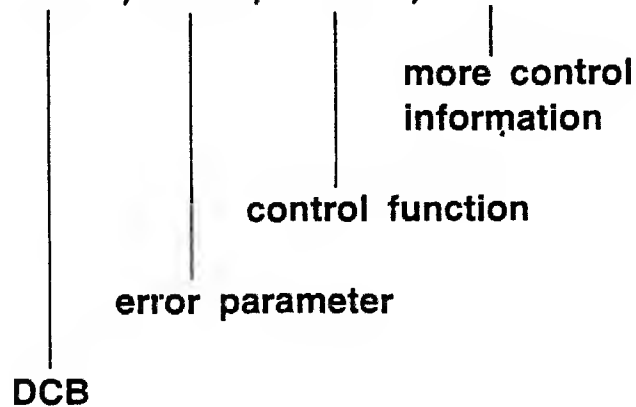
lu number of a device to be accessed
as if there were a type Ø file

CONTROLLING NON-DISC DEVICES

•CALL FCONT•

To control a device through its type **O** file or to control a device opened via **OPENF**, use the **FCONT** routine.

CALL FCONT (IDCB, IERR, ICON1, ICON2)



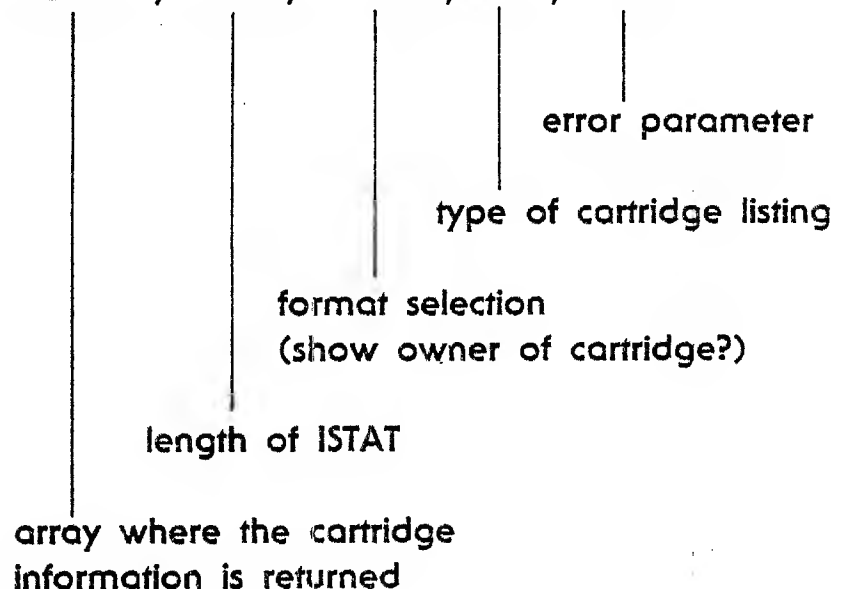
LISTING MOUNTED CARTRIDGES

•CALL FSTAT•

The FSTAT routine returns information about

- the cartridges accessible by your session
- all cartridges currently mounted

CALL FSTAT (ISTAT, ILEN, IFORM, IOP, IADD)



- Reading or writing random records may be done directly in the READF (EREAD) or WRITF (EWRIT) calls.

CALL WRITF (IDCB, IERR, IBUF, IL, NUM)
or CALL READF (IDCB, IERR, IBUF, IL, LEN, NUM)

- positive record number to read or write
- negative number of records to backspace before the read or write

- The file may be positioned to (but not accessed) any random record by calling POSNT (or EPOSN).

CALL POSNT (IDCB, IERR, NUR, IR)

- forward space or backspace a relative number of records
- position to an absolute record

RANDOMLY ACCESSING SEQUENTIAL (TYPE ≥ 3) FILES

- **Type 3 and above files may be positioned to any random record by calling POSNT (or EPOSN).**

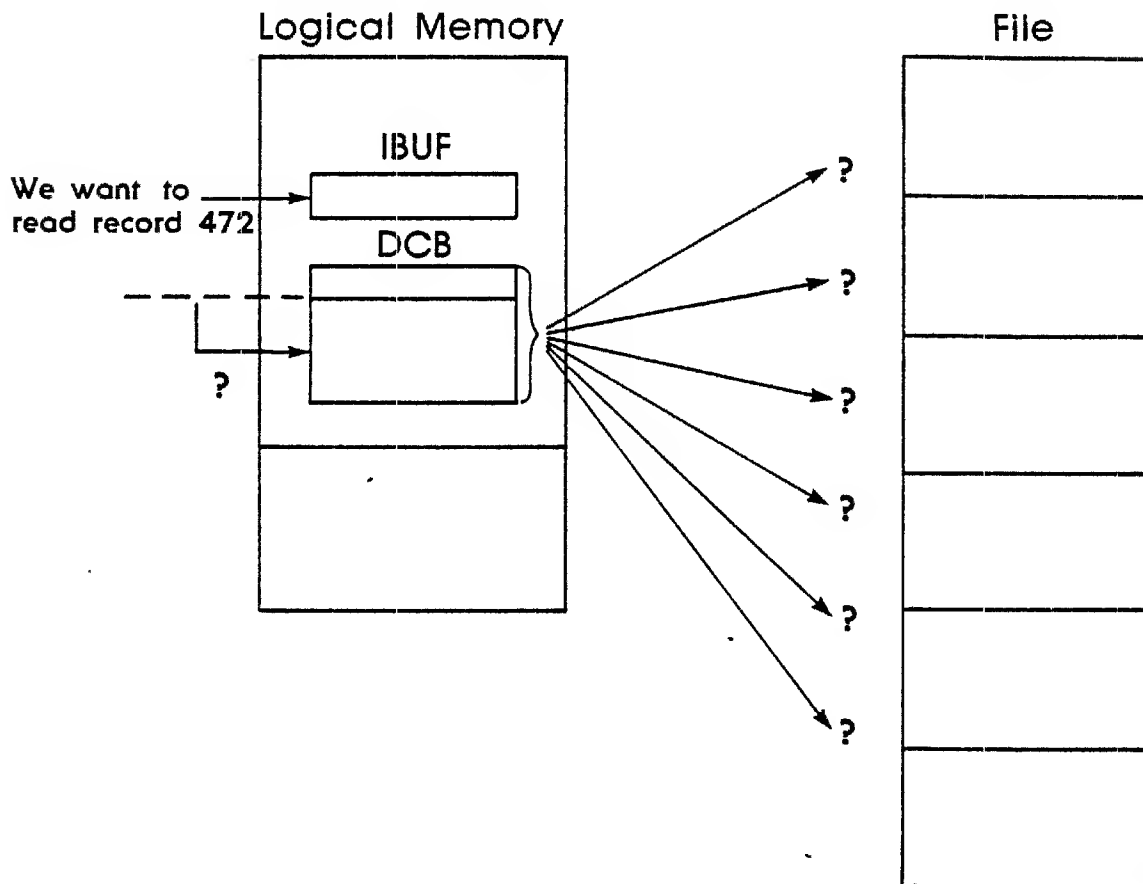
CALL POSNT (IDCB, IERR, NUR, IR)

Positioning is done by READING SEQUENTIAL RECORDS until the desired record is found.

- **Type 3 and above files may be accessed in a truly random fashion if you build a table of record locations in your program and then use this table whenever a record is to be accessed. The routines LOCF (or ELOCF) and APOSN (or EAPOS) allow you to do this.**

ABSOLUTE RECORD LOCATIONS?

What do you need to know about a given record in order to access that record without doing sequential reads?



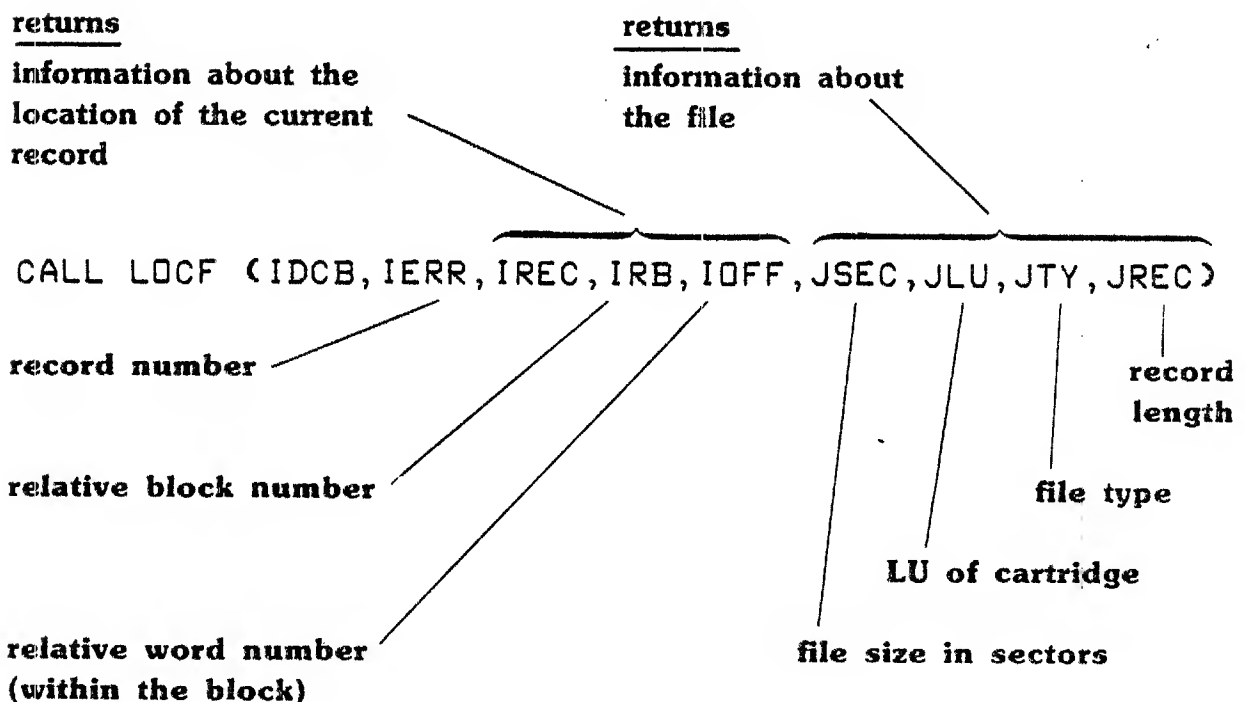
1. which "window" in the file contains the record
2. the location of the record within that "window"

OBTAINING ABSOLUTE RECORD LOCATIONS

•CALL LDCF•
(CALL ELDCF)

These routines return information about the

- size and location of your file.
- location of the record at which the file is currently positioned.



POSITIONING A FILE TO AN ABSOLUTE RECORD LOCATION

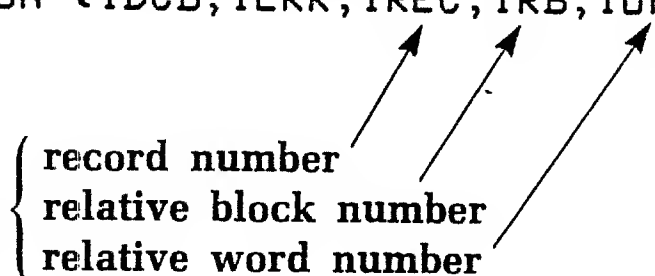
•CALL APOSN•
(CALL EAPOS)

These routines use — the record number
— the relative block number
— the relative word number

to position a file at a specified record.

CALL APOSN (IDCB, IERR, IREC, IRB, IDFF)

obtained from
LOCF (or ELOCF) { record number
relative block number
relative word number



2

USING YOUR RTE SYSTEM



SECTION

A	INTRODUCTION TO FMGR	2-3
B	INTRODUCTION TO PROGRAM DEVELOPMENT IN RTE	2-15

2A. INTRODUCTION TO FMGR

Remember, when you successfully log on, the system runs a copy of the program FMGR at your terminal.

FMGR accepts commands which allow you to use your RTE system.

:
↑
waiting for a command

FMGR uses three devices to perform its functions.

INPUT DEVICE

FMGR accepts commands from the input device.

LIST DEVICE

FMGR outputs the results of certain commands to the list device.

LOG DEVICE

FMGR outputs error messages and accepts corrective commands from the log device.

CAPABILITY LEVELS FOR FMGR COMMANDS

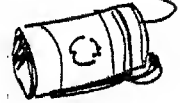
			CAPABILITY LEVEL					
EX HE	SY	TR		1				
AC	MC	TE			10			
CL	ME	WH						
DC	*SL	??						
DL	SM	**						
LI								
AN	DP	RN			20			
CN	DU	ST						
CO	LL	SV						
CR	PK							
CT	PU							
CS								
+OF	RT	SL				30		
RP	RU	SP						
CA	PA	SE					40	
IF	LO							
	SL							50
IN								
OF								60

* Single parameter only

+ Program must be under session's control



MESSAGES



Users can send messages back and forth between each other. RTE keeps a MESSAGE FILE for each of its users.

:SM, user.group, filename, message string

user receiving
the message

file containing a
message

character string
to be sent as a
message

USER'S
MESSAGE
FILE

:ME [lu, filename [, clear]]

where the message(s)
are to be displayed

save (0) or clear (1)
the message

WHAT LU's CAN YOU ADDRESS?

The FMGR SL command will display the session LU's which you can use and their corresponding system LU's.

:SL[,lu]



if specified, displays the system LU of the specified session LU

for example,

```
:SL
SLU 1=LU # 21 = E 5
SLU 2=LU # 2 = E 1
SLU 3=LU # 3 = E 1 S 6
SLU 4=LU # 4 = E 5 S 1
SLU 5=LU # 5 = E 5 S 2
SLU 6=LU # 6 = E 6
SLU 7=LU # 10 = E21
SLU 8=LU # 8 = E 8
SLU 25=LU # 25 = E 1 S16
SLU 28=LU # 28 = E 1 S 1
SLU 38=LU # 38 = E 1 S 2
SLU 47=LU # 47 = E 1 S11
:
```

or

```
:SL,1
SLU 1=LU # 21 = E 5
:
```

The SL command directs its output to the log device.

WHAT DISC CARTRIDGES CAN YOU ACCESS?



The FMGR CL command prints a list of the disc cartridges which you can access.

```
:CL
  LU  LAST TRACK  CR  LOCK  P/G/S

    02      00255   00002          S
    03      00255   00003          S
    28      00149      SP          S
```

* Access to LU 2 and LU 3 is read only unless you are the system manager.

The CL command directs its output to the list device.

WHAT FILES ARE STORED ON THIS DISC CARTRIDGE?

To see what files are stored on a particular cartridge, use the FMGR DL command to list the contents of that cartridge's file directory.

:DL[,cartridge]

↑
the LU (negative) or CRN (positive) of the
cartridge whose file directory is to be listed

for example,

```
:DL,-25
CR=01500
ILAB=DC0021 NXTR= 00000 NXSEC=076 #SEC/TR=096 LAST TR=00050 #DR TR=01
```

NAME	TYPE	SIZE/LU	OPEN TO
------	------	---------	---------

FILE1	00003	00024	BLKS
DATAF	00004	00010	BLKS
&PRDG	00003	00004	BLKS

```
:DL,1500
CR=01500
ILAB=DC0021 NXTR= 00000 NXSEC=076 #SEC/TR=096 LAST TR=00050 #DR TR=01
```

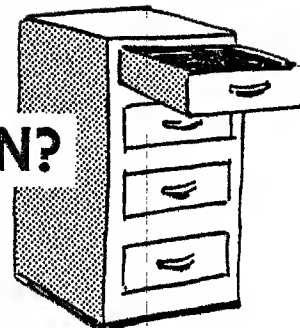
NAME	TYPE	SIZE/LU	OPEN TO
------	------	---------	---------

FILE1	00003	00024	BLKS
DATAF	00004	00010	BLKS
&PRDG	00003	00004	BLKS

:

The DL command directs its listing to the list device.

WHAT DOES THIS FILE CONTAIN?



The FMGR LI command will display the contents of a specified file.

:LI, filename



the name of a file whose contents are to be listed

for example,

```
:LI,&PRG
&PRG  T=00003 IS ON CR SS  USING 00004 BLKS R=0000

0001  FTN4,L
0002      PROGRAM SFAVG( ), FORTRAN PROGRAM, CALCULATE INTEGER AVERAGE
0003  C
0004      WRITE(1,101)
0005  101  FORMAT('THIS PROGRAM CALCULATES INTEGER AVERAGES,')
0006      WRITE(1,102)
0007  102  FORMAT('WHEN ASKED, ENTER AN INTEGER > 0.')
```

0008 WRITE(1,103)

0009 103 FORMAT('ENTER 0 TO TERMINATE INPUT')//

0010 C

0011 ICT = 0

0012 ISUM = 0

0013 C

.

.

.

The LI command also lists on the list device.

CHANGING THE LIST DEVICE

You can change your list device with the FMGR LL command.

:LL, 1u



an LU which is to become the new list device

for example,

```
:CL
  LU  LAST TRACK  CR  LOCK  P/G/S
02    00255    00002      S
03    00255    00003      S
28    00149      SP      S
```

```
:LL,6
:CL
:LL,1
:CL
  LU  LAST TRACK  CR  LOCK  P/G/S
02    00255    00002      S
03    00255    00003      S
28    00149      SP      S
:
```

RUNNING PROGRAMS

Use the FMGR RU command to run a program. FMGR requests RTE to schedule the program for execution and then waits for the program to complete before issuing another prompt.

:RU, program name

↑
name of a program to be run

for example,

```
:RU,SFAVG
THIS PROGRAM CALCULATES INTEGER AVERAGES,
WHEN ASKED, ENTER AN INTEGER > 0.
(ENTER 0 TO TERMINATE INPUT)
```

```
INPUT AN INTEGER VALUE: 482
INPUT AN INTEGER VALUE: 412
INPUT AN INTEGER VALUE: 437
INPUT AN INTEGER VALUE: 0
```

```
YOU ENTERED      3 VALUES
```

```
THE SUM WAS 1331
```

```
THE AVERAGE WAS 443
:
```

FMGR ERROR MESSAGES

FMGR uses error messages to report

- information
- input errors or illegal commands

FMGR error messages are of the form:

FMGR nnn

↑
a positive or negative error number

for example,

```
:LI,AFILE  
FMGR-006  
:DK,-25  
FMGR 010  
DK,?  
:SE,24  
FMGR 046  
:
```


WHAT DOES THAT ERROR NUMBER MEAN?

There are two ways to get information about a FMGR error.

LEVEL 1 Use the FMGR ?? command to see a one-line explanation of a FMGR error.

:??[,error number]

↑
*error number to be explained; the explanation
is printed on the list device*

for example,

```
:LI,AFILE
FMGR-006
:??
FMGR -06 FILE NOT FOUND.
:??,10
FMGR 010 INPUT ERROR
:
```

"HELP

The file "HELP contains explanations about

- system errors
- FMGR errors
- LOADR errors
- etc.

A second way to get information about a FMGR error is to consult the "HELP file.

LEVEL II Use the FMGR HE command to see the "HELP file explanation about one of these errors.

:HE, [, keyword [, lu]]

lu where the explanation is printed

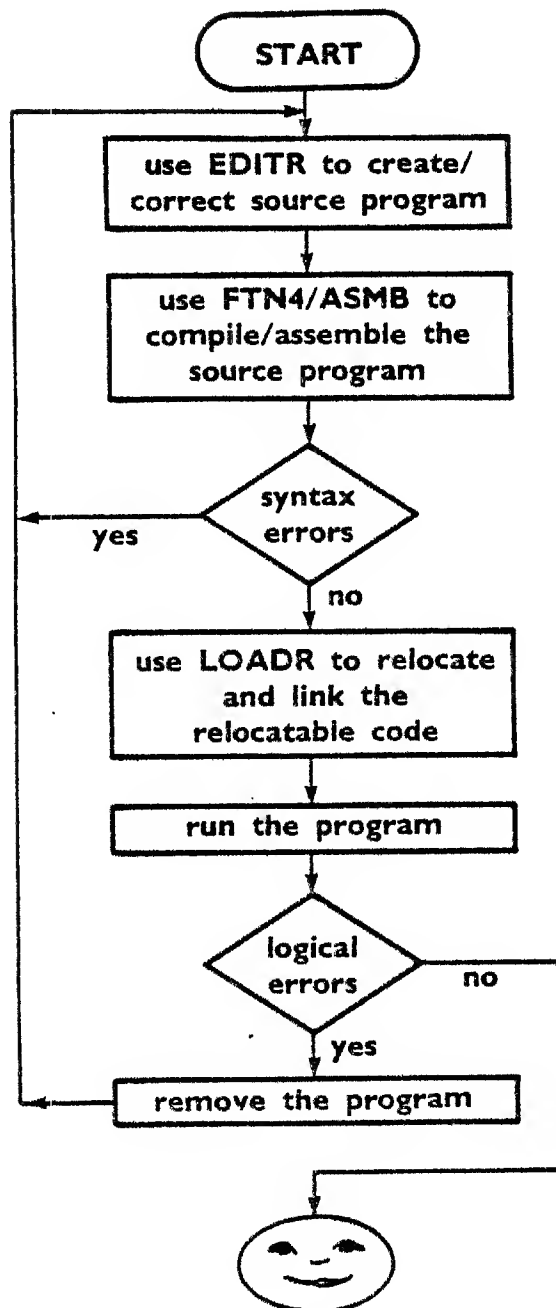
keyword of the error to be explained, i.e. FMGR-006

```
:LI,AFILE
FMGR-006
:HE
FMGR-006
FILE NOT FOUND
AN ATTEMPT WAS MADE TO ACCESS A FILE THAT CANNOT BE FOUND  CHECK THE
FILE NAME OR THE CATRIDGE REFERENCE.
```

```
:HE,IO 12
HELP PROGRAM ERROR  HELP0002
KEYWORD NOT FOUND
      KEYWORD WAS NOT FOUND IN THE HELP FILE.  IF THE
      KEYWORD WAS SPECIFIED IN THE COMMAND, CHECK IT.
:HE,IO12
IO12
AN I/O REQUEST SPECIFIED A LOGICAL UNIT NOT DEFINED FOR USE BY
THIS SESSION.  THE "SL" COMMAND WILL REPORT ALL LOGICAL UNITS
AVAILABLE TO YOUR SESSION.
```

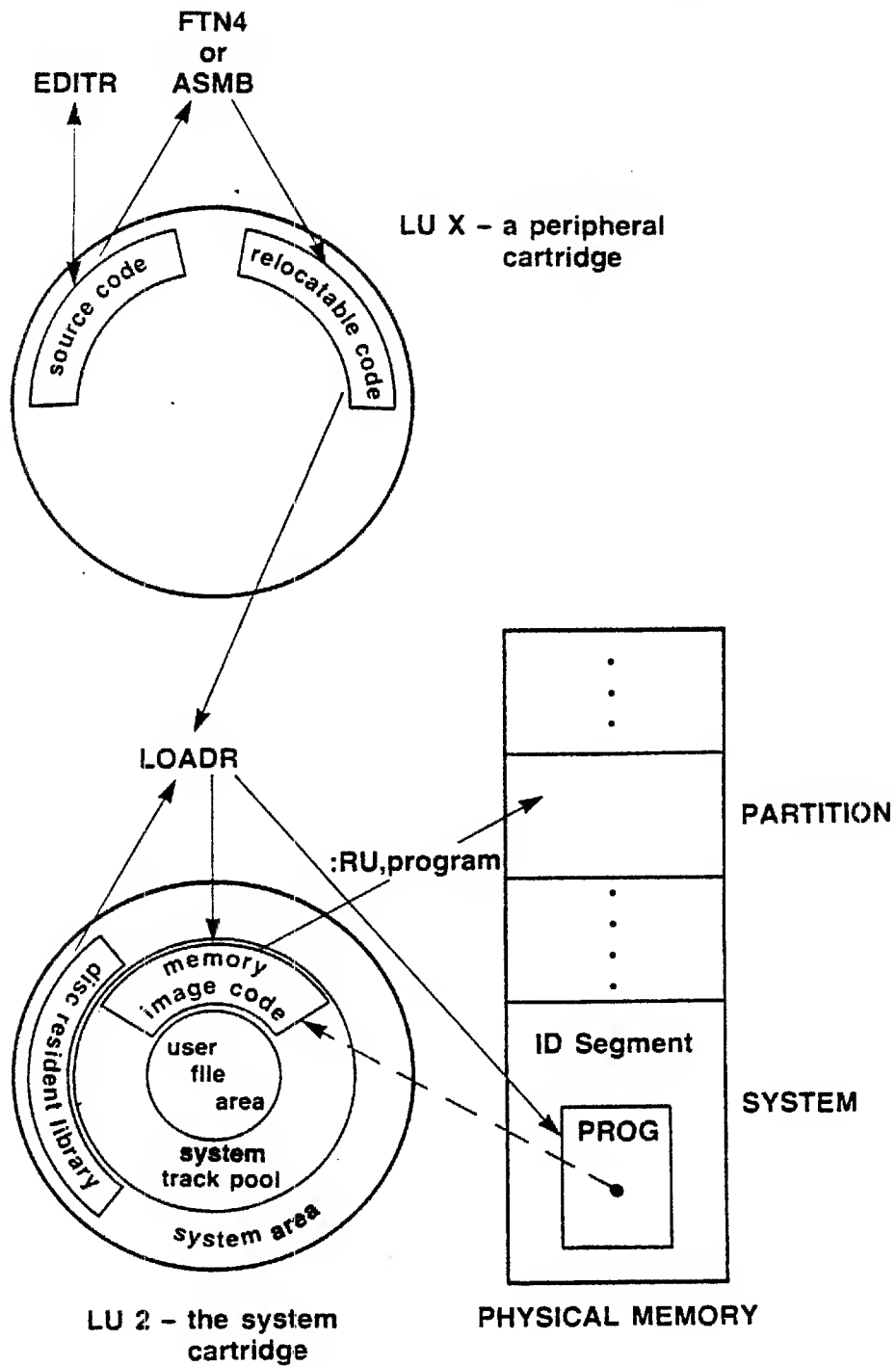
:

2B. INTRODUCTION TO PROGRAM DEVELOPMENT IN RTE

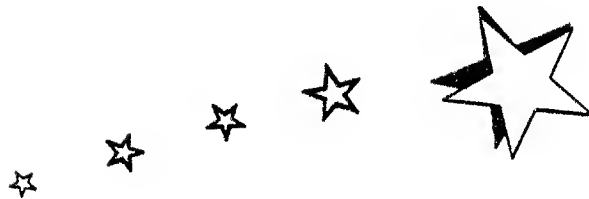


1. The source program consists of your FORTRAN or Assembly Language statements.
2. The FORTRAN compiler or RTE Assembler processes the source statements and creates the relocatable code.
3. LOADR relocates and links your program's routines and any needed library routines and fills in an ID segment in memory and creates the memory image code on disc for your program.
4. When you run your program, the memory image code is loaded from disc into memory and then executed.
5. If you need to correct your program, you will need to release its ID segment and disc tracks before loading a new version.

PROGRAM DEVELOPMENT IN RTE



USING EDITR



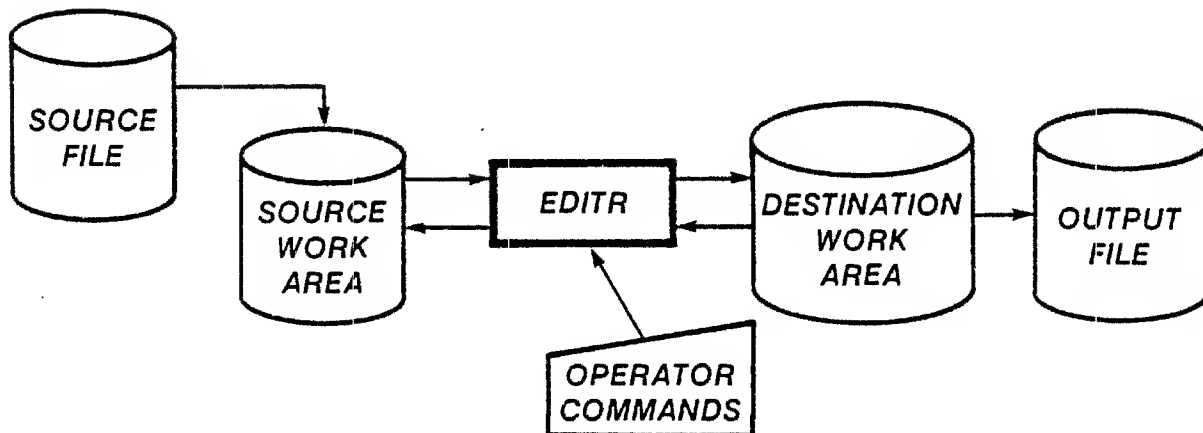
The EDITR program provides an easy means to

- **create an ASCII file containing a source program, data or text.**
- **modify an existing ASCII file.**
- **append (merge) files together.**

EDITR WORK AREAS

EDITR automatically allocates two work areas on the disc

- source work area
- destination work area



The EDITR commands are used to modify lines of ASCII data in the source work area. EDITR automatically moves the corrected lines into the destination work area.

RUNNING EDITR

To invoke EDITR, use the RU command.

```
:RU,EDITR  
SOURCE FILE?  
/
```



The slash (/) is EDITR's prompt for input. Here, EDITR is asking you for the name of the ASCII file to be edited. Enter

- a file name; EDITR will copy the file into its source work area
- zero (0) to indicate you want to create a new file (use a fresh work area)
- colon (:) to terminate EDITOR

for example,

```
:RU,EDITR  
SOURCE FILE?  
/&PROG  
FTN4,L  
/
```

← EDITR prints the first line
← and then prompts for an
EDITR command

TYPES OF EDITR COMMANDS

DISPLAY

Control the pending line or supply information

LINE EDITS

Edit one line at a time

CHARACTER EDITS

Edit individual characters within a line

PATTERN EDITS

Edit pattern sequences and blocks of text



AN EDITING EXAMPLE

Suppose file &PROG contains this source program.

```
:LI,&PROG
&PROG  T=00003 IS ON CR SS   USING 00004 BLKS R=0000

0001  FTN4,L
0002  PROGRAM SFAVG
0003  C
0004          WRITE(1,101)
0005  101    FORMAT('THIS PROGRAM CALCULATES INTEGER AVERAGES,')
0006  102    FORMAT('WHEN ASKED, ENTER AN INTEGER > 0.')

```

:

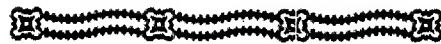
DISPLAY COMMANDS

These commands control the pending line or supply information:

<u>command</u>	<u>description</u>
/P	list the pending line
/Ln	list the pending line and the next “n” lines, making the last line the new pending line
//	advance the pending line by one
/+n	advance the pending line by “n” lines (/ + 1 is the same as /+ or //)
/^n	back up “n” lines
/N	display the line number of the pending line
/n	make line “n” the pending line
/HL	display column numbers

(and others)

EXAMPLES OF DISPLAY COMMANDS



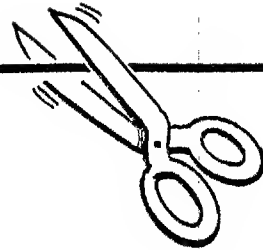
```
:RU,EDITR
SOURCE FILE?
/&PROG
  FTN4,L
/P
  FTN4,L
/LS
  FTN4,L
PROGRAM SFAVG
C
      WRITE(1,101)
101   FORMAT(''THIS PROGRAM CALCULATES INTEGER AVERAGES, '')
102   FORMAT(''WHEN ASKED, ENTER AN INTEGER > 0.'')
/^
101   FORMAT(''THIS PROGRAM CALCULATES INTEGER AVERAGES, '')
/1
  FTN4,L
```

EXAMPLES OF LINE EDIT COMMANDS

```
/1
  FTN4,L
//
  PROGRAM SFAVG
/HL
  /////1/////2/////3/////4/////5/////6/////
'8
/R      PROGRAM SFAVG
/P
      PROGRAM SFAVG
//
  C
//
      WRITE(1,101)
//
  101  FORMAT('THIS PROGRAM CALCULATES INTEGER AVERAGES,')
/      WRITE(1,102)
/P
      WRITE(1,102)
//
  102  FORMAT('WHEN ASKED, ENTER AN INTEGER > 0.')
```

```
//
      WRITE(1,103)
//
  103  FORMAT(
/-
  103  FORMAT('(ENTER 0 TO TERMINATE INPUT)')
```

LINE EDIT COMMANDS



These commands edit one line at a time:

<u>command</u>	<u>description</u>
/R text	Replace the pending line with "text"
/I text	Insert "text" before pending line
/Δ text	Insert "text" after pending line, make "text" new pending line
/-n	Delete "n" lines, starting with the pending line
(and others)	

(note: Δ is a blank)



CHARACTER EDIT COMMANDS

These commands edit individual characters within a line:

<u>command</u>	<u>description</u>
/P////text	Replace the characters in the pending line with the characters in ``text'', character by character according to position. The slash (/) represents a placeholder, to preserve existing characters.
/P////text ① or ⑤	Insert the characters in ``text'' into the pending line.
/P////xxx ③	Cancel (delete) characters in the pending line corresponding to the placeholder ``x''s.
/P/// ④	Truncate the pending line.

(and others)

(Note: ○ represents a CONTROL character)



EXAMPLES OF CHARACTER EDIT COMMANDS



```
103  FORMAT(''(ENTER 0 TO TERMINATE INPUT)''/)  
/+3  
      ISUM - 0  
/P/////////=  
      ISUM = 0  
//  
  C  
//  
  10  WRITE(104)  
/P/////////1,  
  10  WRITE(1,104)  
/+2  
      READ(1,*9) IVAL  
/P/////////X  
      READ(1,*) IVAL  
/+4  
      GOTO 101  
/P/////////  
      GOTO 10
```

Diagram annotations:

- Circle S with an arrow pointing to the first '1' in the line `/P/////////1,`
- Circle C with an arrow pointing to the 'X' in the line `/P/////////X`
- Circle T with an arrow pointing to the first character in the line `/P/////////`



PATTERN EDIT COMMANDS



These commands edit pattern sequences of characters:

<u>command</u>	<u>description</u>
/G old text/new text	Replaces each occurrence of "old text" with "new text" in the pending line.
/Fx text	Search for the next line which contains "text" and make that line the pending line.
a control option	
— a slash	
— an escape	
— nothing	
(and other commands)	

EXAMPLES OF PATTERN EDIT COMMANDS

```
/+3
    IAVG = ISUM/ICT
/GIAVG/      IAVG
            IAVG = ISUM/ICT
/F/WRIT
            WRITE(1,105) ICT
/F/WRIT(
            WRIT(1,107) IAVG
/GWRIT/WRITE
            WRITE(1,107) IAVG
//
    107   FORMAT(''THE AVERAGE WAS '' ,I5)
//
            END
//
    END$'
/
```



TERMINATING EDITR

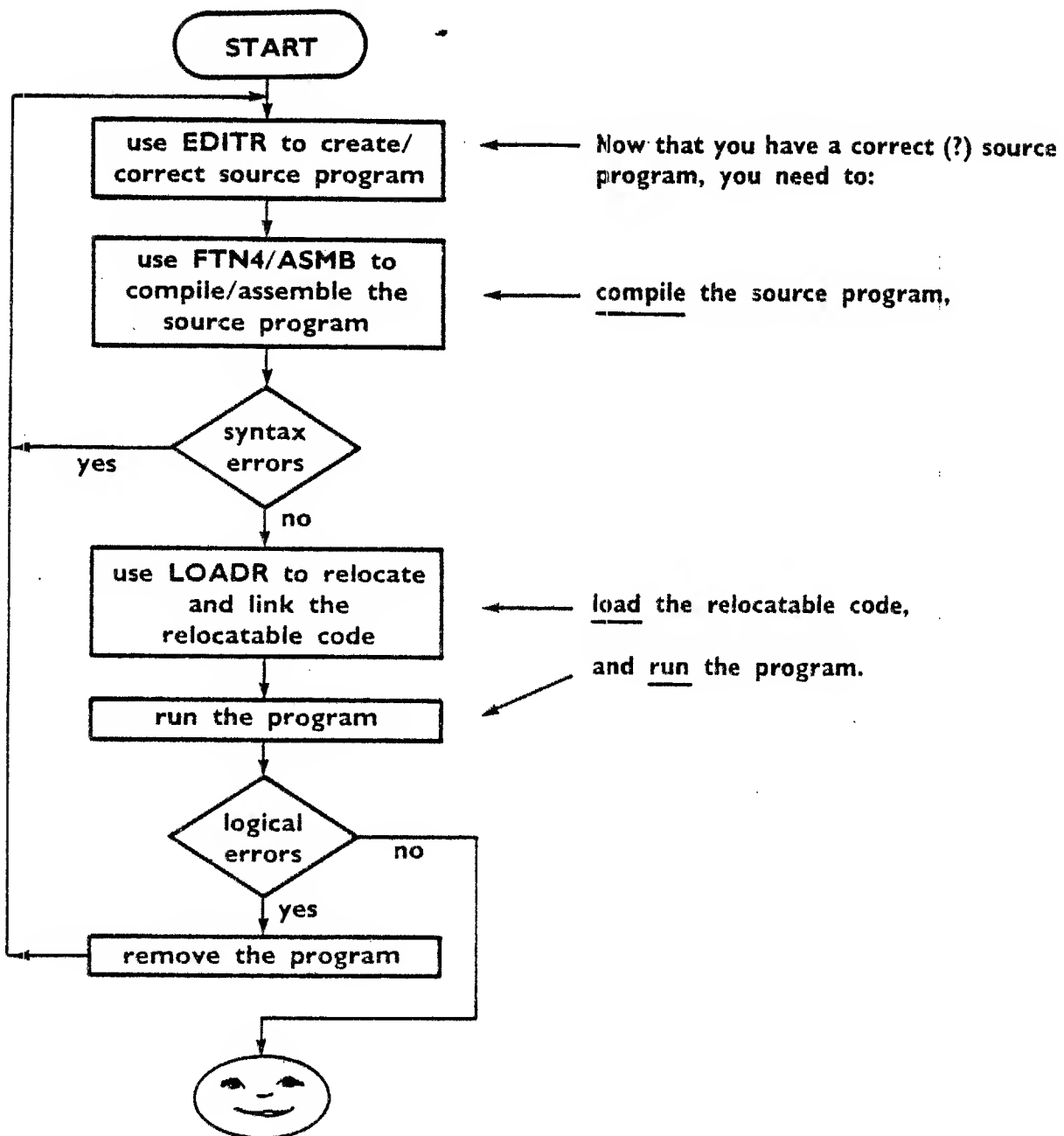
After editing, you must tell the EDITR to store its destination work area in a file you specify. Use one of these commands:

/EC filename	End and Create a new file called "filename"
/ER filename	End and Replace the contents of file "filename"
/ER	End and Replace the contents of the file you named as the source file.

To abort the EDITR without saving your work or altering any existing files, use the command:

/A	Abort the EDITR without saving the destination work area.
-----------	--

COMPILING, LOADING AND RUNNING YOUR PROGRAM



FTN4 — THE FORTRAN IV COMPILER



The RTE FORTRAN IV compiler takes FORTRAN IV source statements as input and outputs:


1. relocatable code
2. a program listing
3. diagnostic error messages if errors occurred

The relocatable code is in a format suitable for the RTE LOADR.


Run FTN4 by —

:RU,FTN4, source file name, list lu, relocatable file name


contains the
source program



where to print
the listing



where to put the
relocatable code



RUNNING FTN4 — AN EXAMPLE

:RU,FTN4,&PROG,1,%PROG

PAGE 0001 FTN. 8:53 AM THU., 31 MAY , 1979

```
0001 FTN4,L
0002      PROGRAM SFAVG
0003 C
0004      WRITE(1,101)
0005 101  FORMAT('THIS PROGRAM CALCULATES INTEGER AVERAGES,')
0006      WRITE(1,102)
0007 102  FORMAT('WHEN ASKED, ENTER AN INTEGER > 0.')
```

0008 WRITE(1,103)

0009 103 FORMAT(' (ENTER 0 TO TERMINATE INPUT) ')

0010 C

0011 ICT = 0

0012 ISUM = 0

0013 C

0014 10 WRITE(1,104)

0015 104 FORMAT('INPUT AN INTEGER VALUE: _')

0016 READ(1,*) IVAL

0017 IF(IVAL .LE. 0) GOTO 99

0018 ICT = ICT + 1

0019 ISUM = ISUM + IVAL

0020 GOTO 10

0021 C

0022 99 IF(ICT .LE. 0) GOTO 10

0023 IAVG = ISUM/ICT

0024 WRITE(1,105) ICT

0025 105 FORMAT('YOU ENTERED ',IS,' VALUES')

0026 WRITE(1,106) ISUM

0027 106 FORMAT('THE SUM WAS ',IS)

0028 WRITE(1,107) IAVG

0029 107 FORMAT('THE AVERAGE WAS ',IS)

0030 END

FTN4 COMPILER: HP92060-16092 REV. 1913 (790206)

** NO WARNINGS ** NO ERRORS ** PROGRAM = 00211 COMMON = 00000

PAGE 0002 FTN. 8:53 AM THU., 31 MAY , 1979

0031 END*

\$END FTN4: NO DISASTRS NO ERRORS NO WARNINGS

LOADR — THE RTE IV LOADER

The RTE LOADR takes relocatable code as input and outputs:

1. memory-image code in the system track pool area of LU 2 or 3.
2. an ID segment in memory.
3. a load map.
4. diagnostic error messages if any occurred.


Run LOADR by —

`:RU,LOADR,, relocatable file name, list lu`

contains the
relocatable code



where to print
the listing



RUNNING LOADR -- AN EXAMPLE

*

*

*

*

*

:RU,LOADR,,%PROG,1
SFAVG 40042 40364

FMTIO	40365	41663	24998-16002	REV.1926	790417
FMT.E	41664	41664	24998-16002	REV.1901	781107
PNAME	41665	41732	771121	24998-16001	
REIO	41733	42057	92067-16268	REV.1903	790316
FRMTR	42060	45515	24998-16002	REV.1926	790503
.CFER	45516	45573	750701	24998-16001	

4 PAGES RELOCATED	4 PAGES REQ'D	NO PAGES EMA	NO PAGES MSEG
LINKS:BP	PROGRAM:BG	LOAD:TE	COMMON:NC
/LOADR:SFAVG READY AT 2:27 PM THU., 1 MAY, 1980			

/LOADR:SEND

:



RUNNING YOUR PROGRAM

Once you have created the memory image code and filled in an ID segment, you can run your program:

```
:RU,SFAVG  
THIS PROGRAM CALCULATES INTEGER AVERAGES,  
WHEN ASKED, ENTER AN INTEGER > 0.  
(ENTER 0 TO TERMINATE INPUT)
```

```
INPUT AN INTEGER VALUE: 425  
INPUT AN INTEGER VALUE: 413  
INPUT AN INTEGER VALUE: 465  
INPUT AN INTEGER VALUE: 478  
INPUT AN INTEGER VALUE: 436  
INPUT AN INTEGER VALUE: 0
```

```
YOU ENTERED      5 VALUES
```

```
THE SUM WAS  2217
```

```
THE AVERAGE WAS  443  
:
```


REMOVING PROGRAMS

RTE only allows one program for each program name; if you wish to re-load a program which already exists, you must release its ID segment and disc tracks.

The FMGR `OF` command will do this for you.

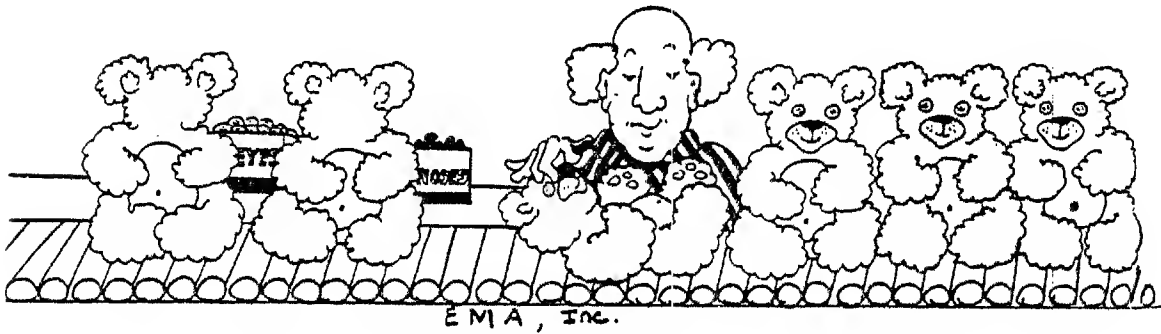
`:OF, program name`

The FMGR `OF` command will:

- terminate the program if it is executing.
- release the program's ID segment and the disc tracks containing its memory image code.

3

RTE ORGANIZATION



SECTION

A	BREAKMODE — INTERACTING WITH RTE	3-3
B	RTE CONCEPTS	3-9
C	TROUBLE SHOOTING	3-29
D	BREAKMODE vs SYSTEM COMMANDS	3-44

3A. BREAKMODE — INTERACTING WITH RTE

Consider this program:

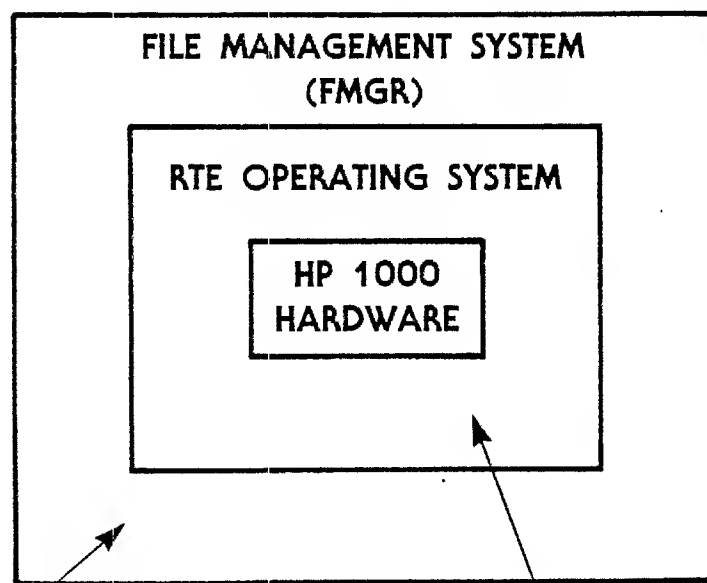
```
                PROGRAM INF
C
10      CONTINUE
        GO TO 10
        END
```

If you run this program,

:RU, INF

← FMGR will wait
for your program
to complete
before issuing
another prompt.

BREAKMODE vs FMGR COMMANDS



You normally operate at this level, using FMGR commands.

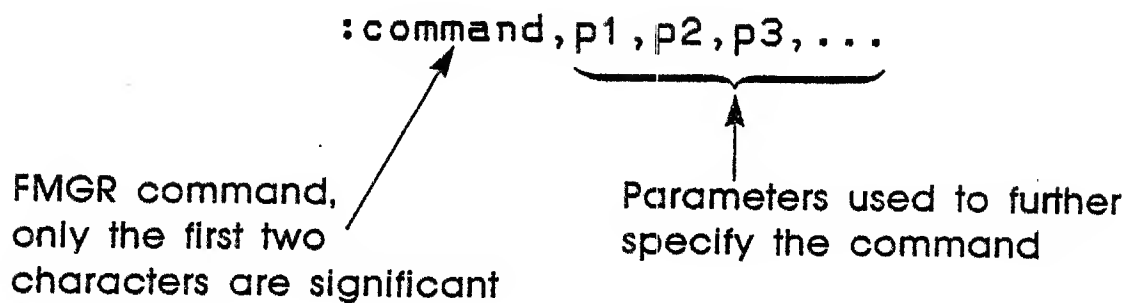
RTE will also accept commands. Enter any breakmode command in response to a breakmode prompt:

S = xx COMMAND?

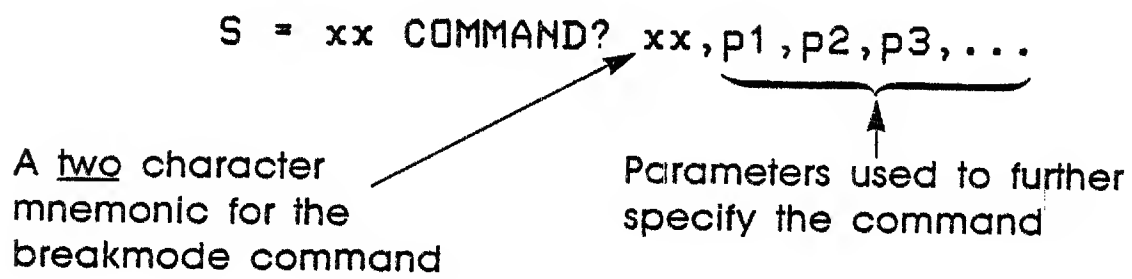
↑
waiting
for
a
breakmode
command

COMMAND FORMATS

FMGR COMMANDS —



BREAKMODE COMMANDS —



EXAMPLES OF BREAKMODE SERVICES

There are several types of breakmode commands:

- **GET STATUS OF SYSTEM**

TI displays current system time

ST displays status of a program

TD displays timeout value of a device

- **CONTROL PROGRAMS AND I/O OPERATIONS**

UP make a peripheral device available

SS suspend a program

OF terminate a program

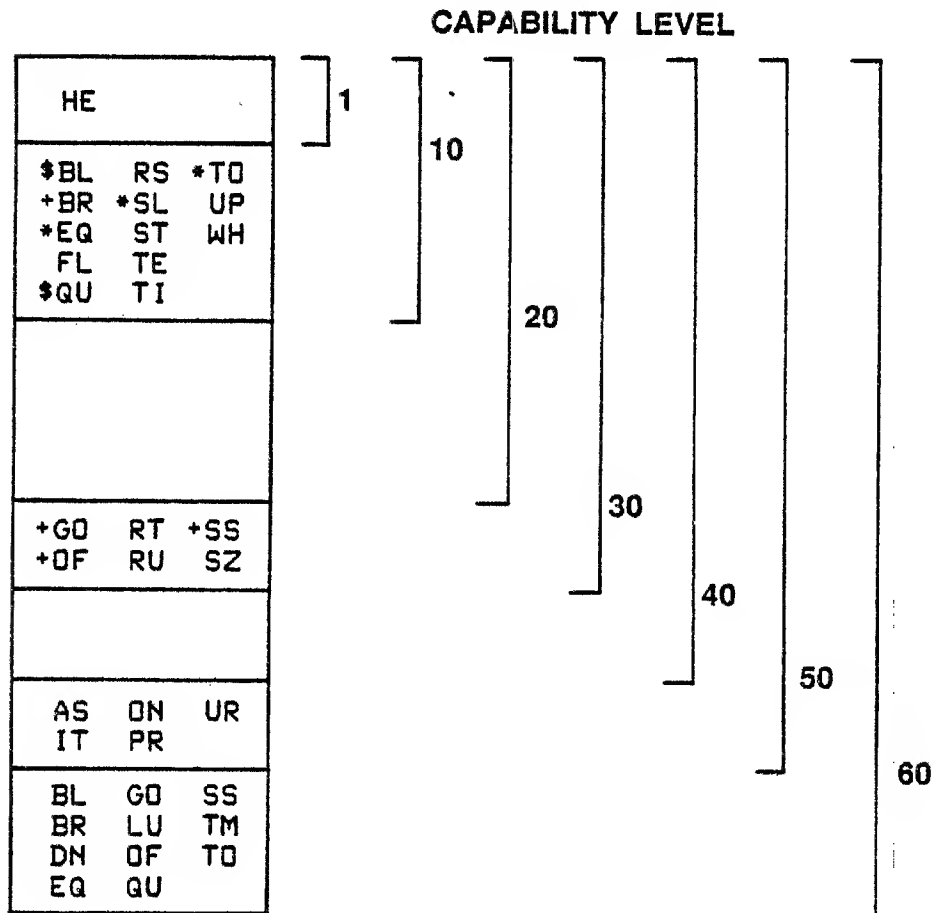
- **ALTER SYSTEM PARAMETERS**

TM set system time

QU change time-slicing parameters

TD change the time-out value of a device

CAPABILITY LEVELS FOR BREAKMODE COMMANDS



- * Single parameter only
- + Program must be under session's control
- \$ No parameters permitted

EXAMPLES OF BREAKMODE SERVICES

There are several types of breakmode commands:

- * GET STATUS OF SYSTEM

- TI - displays current system time
- ST - displays status of a program
- TO - displays timeout value of a device

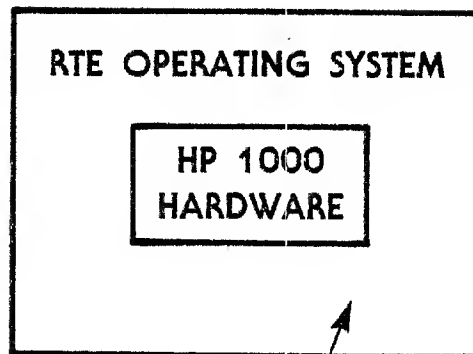
- * CONTROL PROGRAMS AND I/O OPERATIONS

- UP - make a peripheral device available
- SS - suspend a program
- OF - terminate a program

- * ALTER SYSTEM PARAMETERS

- TM - set system time
- QU - change time-slicing parameters
- TO - change the time-out value of a device

3B. RTE CONCEPTS

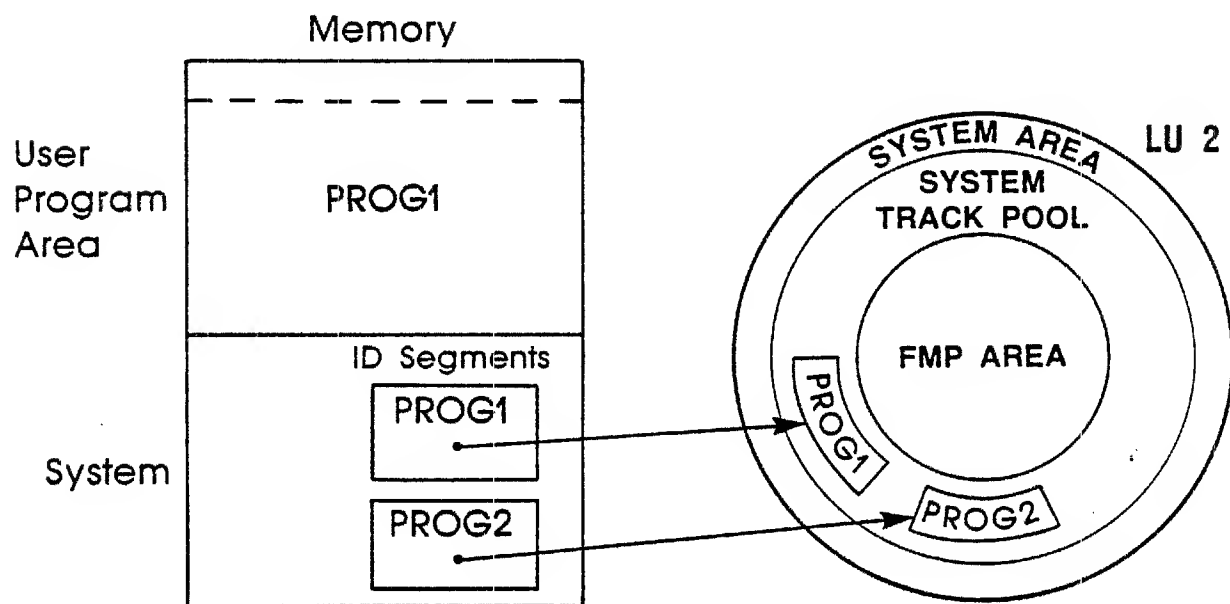


RTE manages system resources.

- MEMORY
- I/O OPERATIONS
- PROGRAMS

MEMORY MANAGEMENT

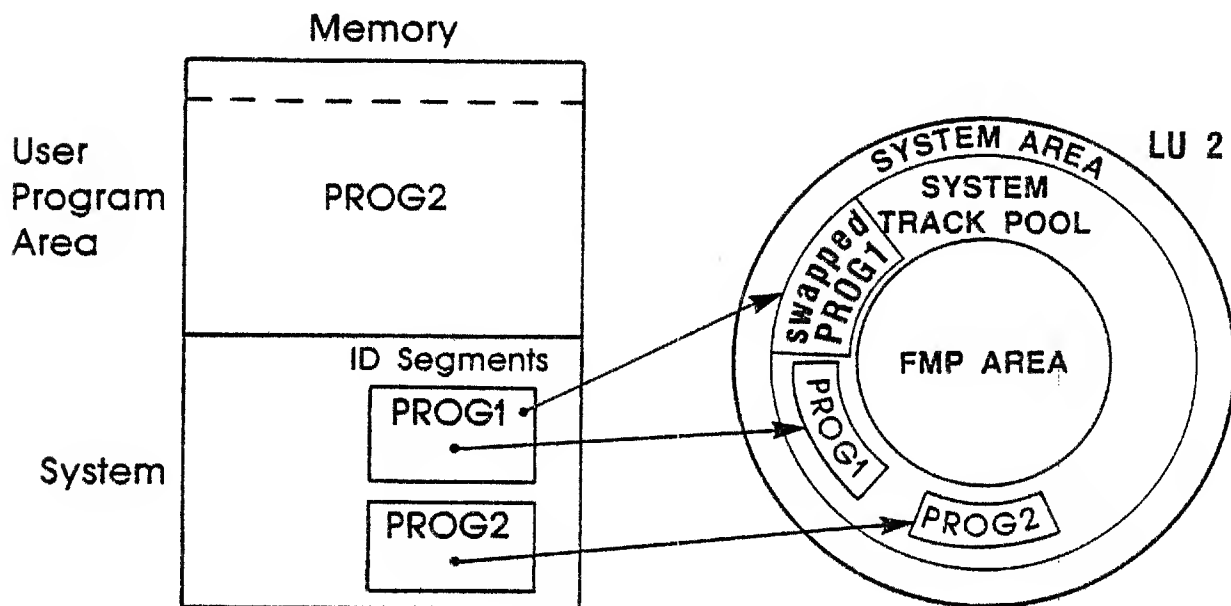
Suppose you have two programs, PROG1 and PROG2. PROG1 is currently executing and so resides in the user program area of memory; PROG2 is waiting its turn for execution.



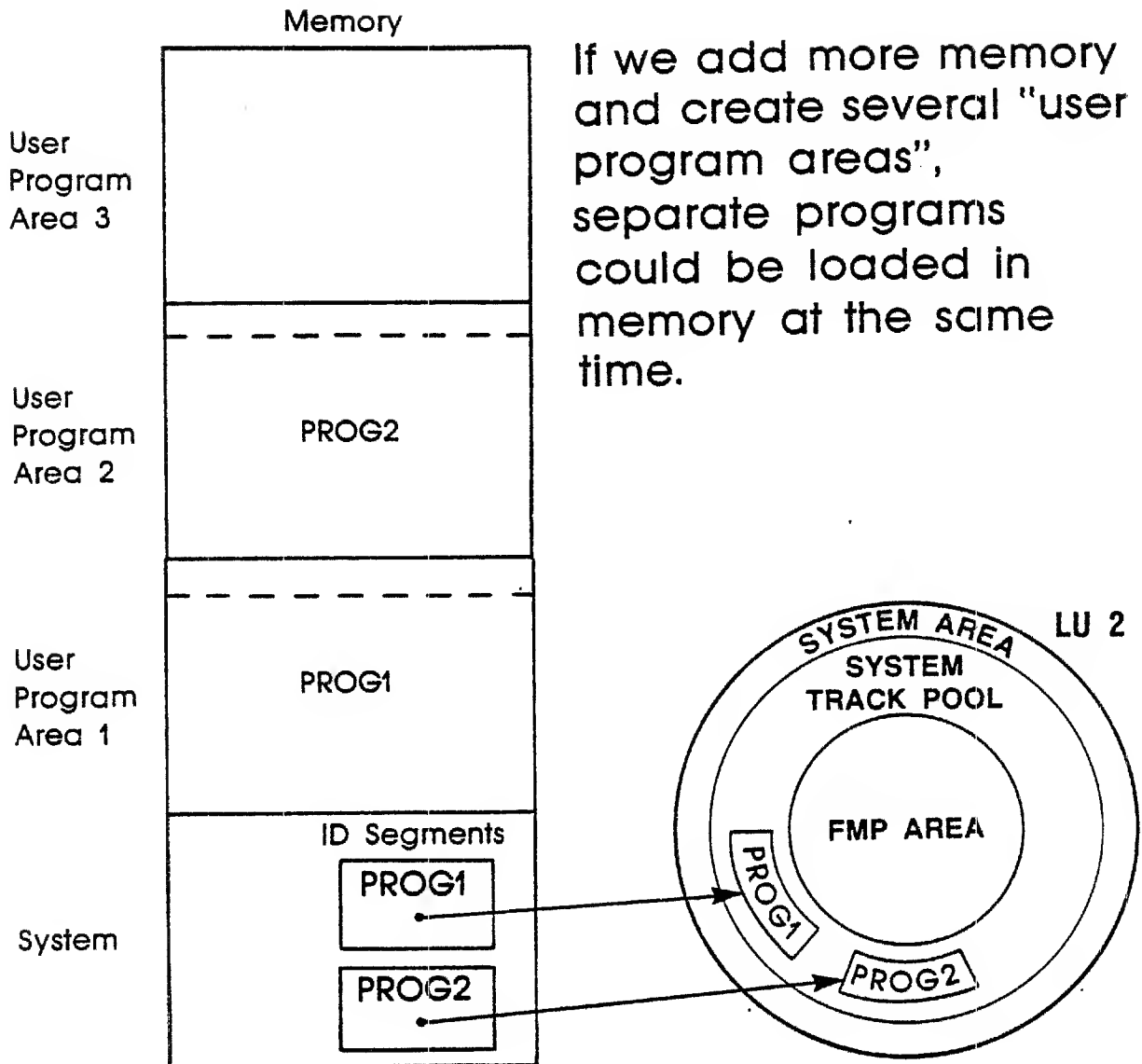
If PROG1 initiates a data transfer, how does the CPU begin execution of PROG2?

SWAPPING

If PROG1 initiates a data transfer, then the CPU can execute PROG2. PROG1 must first be *swapped* out of memory so PROG2 can be loaded into memory and executed.



PARTITIONS



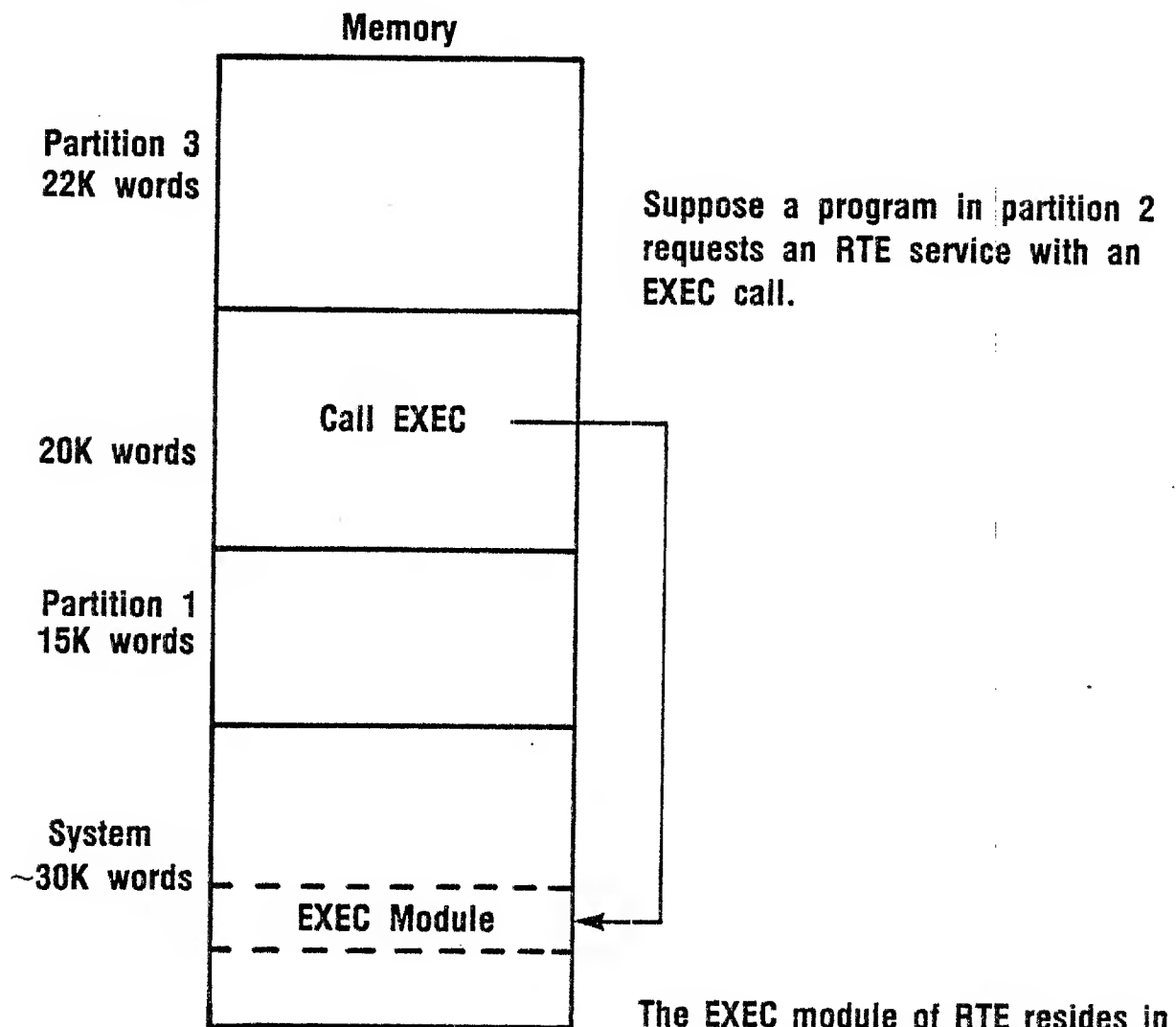
If we add more memory and create several "user program areas", separate programs could be loaded in memory at the same time.

Separate user program areas are called PARTITIONS. Partitions are defined when the RTE system is generated.

TOO MUCH MEMORY?

HP 1000 computers have 15 bits for addressing; hence, you can access 32K words or memory locations.

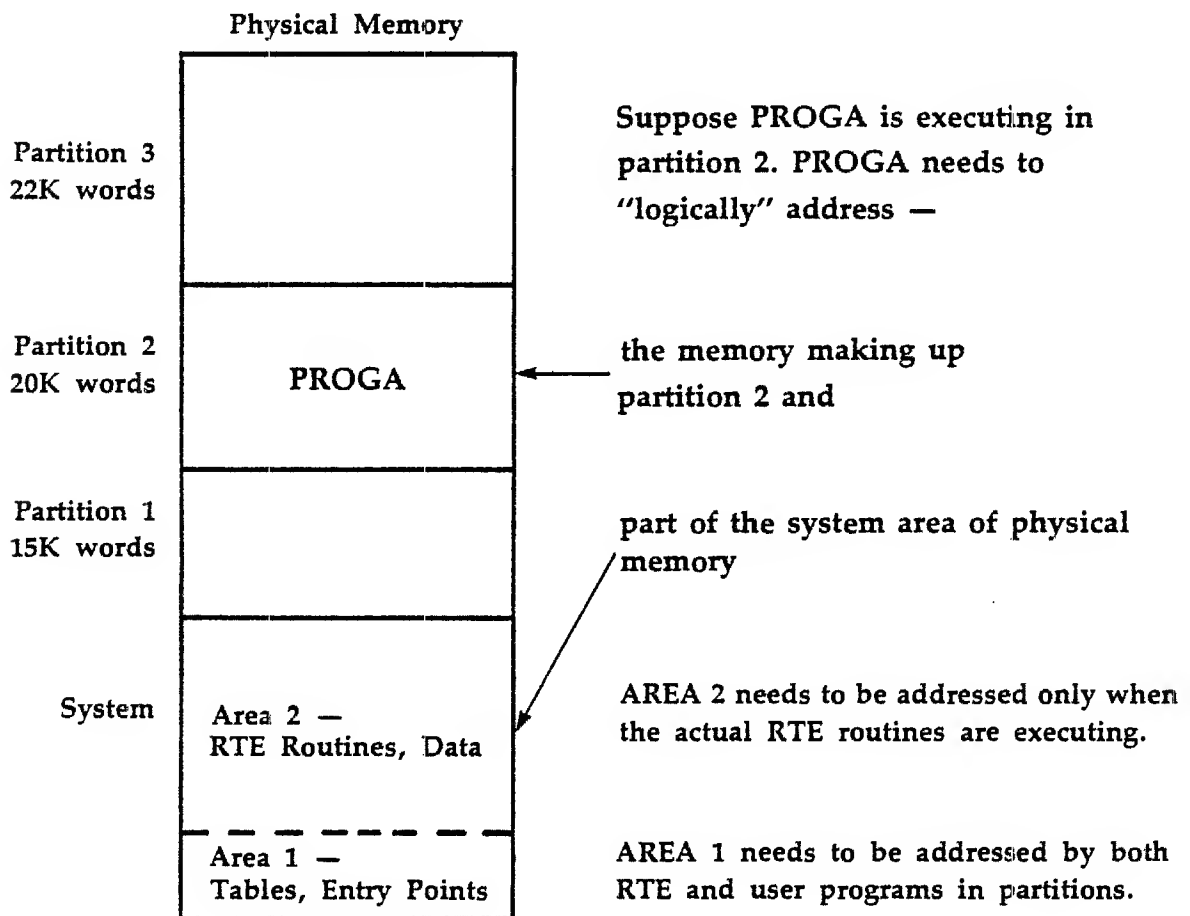
An RTE system with several partitions might have memory sizes as follows:



DYNAMIC MAPPING SYSTEM

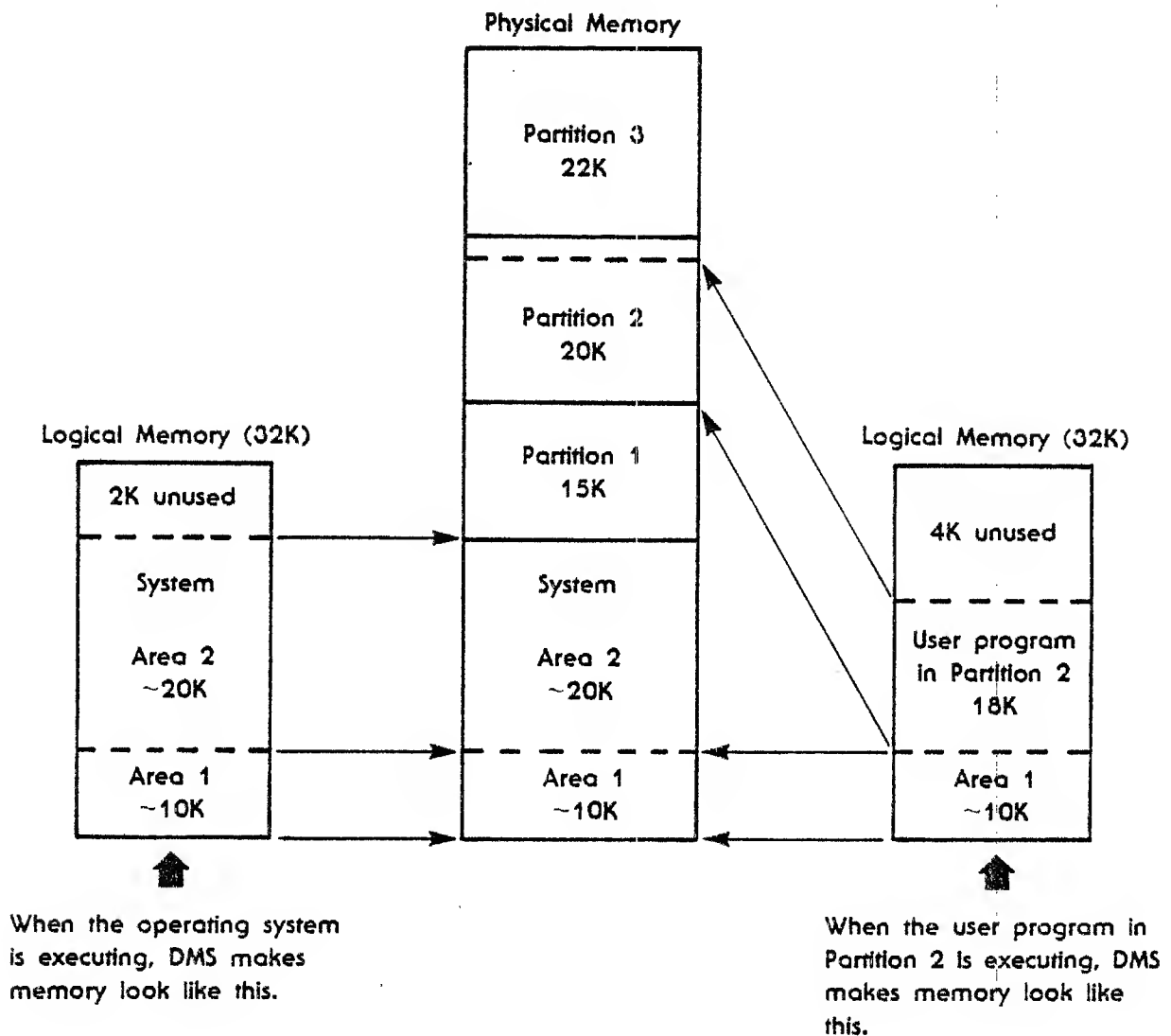
The DYNAMIC MAPPING SYSTEM (DMS) allows an HP 1000 computer to access up to 1024K words of "physical memory" using 15 bits for addressing.

With DMS, the 32K words which your program can "logically" address only needs to include that part of physical memory needed to execute the program.



LOGICAL vs PHYSICAL MEMORY

RTE, in conjunction with DMS, allows access to only those parts of physical memory which are needed by the currently executing program.



BACKGROUND vs REAL-TIME

Partitions may be defined as

BACKGROUND or REAL-TIME PARTITIONS

Programs may be defined as

BACKGROUND or REAL-TIME PROGRAMS

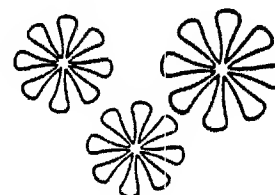
- **BACKGROUND programs will run in BACKGROUND partitions.**
- **REAL-TIME programs will run in REAL-TIME partitions.**
- **If only one type of partition is defined, all programs will run in that type of partition.**
- **Programs may be assigned to a partition; a partition may be reserved for a specific program.**

Both BACKGROUND and REAL-TIME programs reside on the disc, hence the terms:

REAL-TIME DISC RESIDENT PROGRAMS

BACKGROUND DISC RESIDENT PROGRAMS

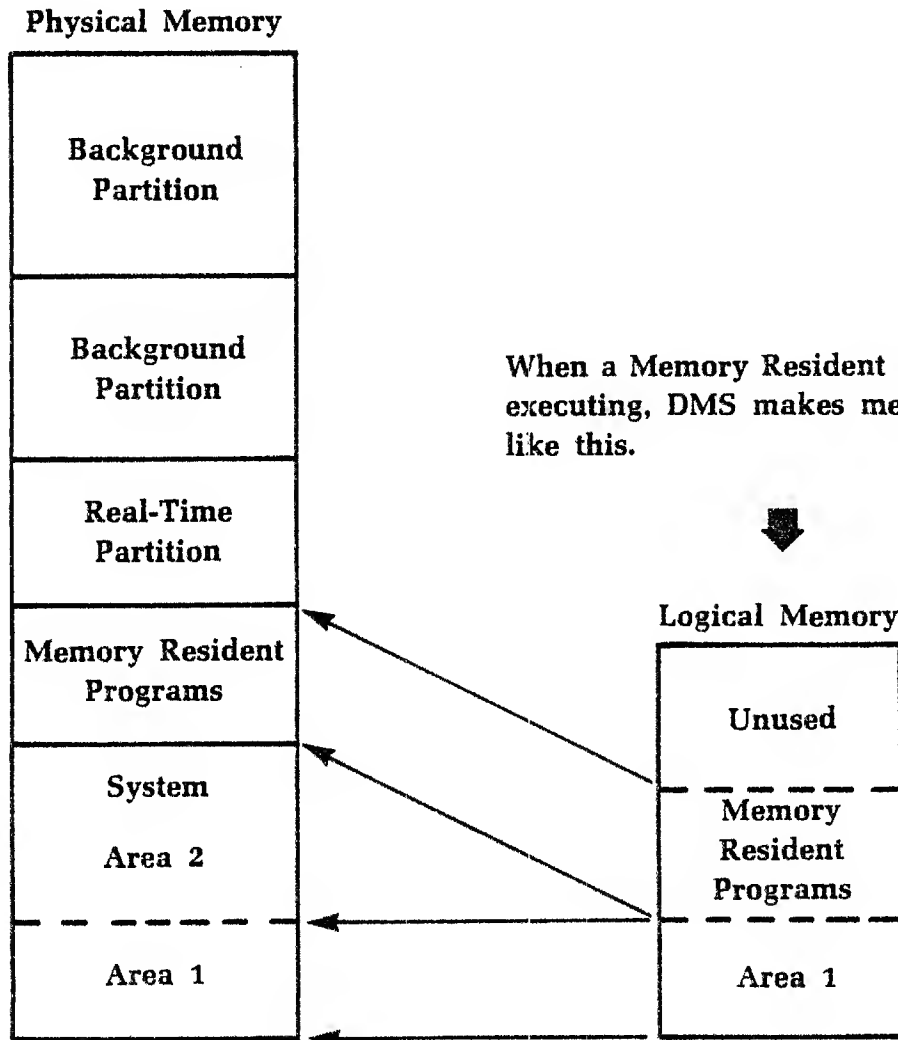
Both types of disc resident programs can be swapped.



MEMORY RESIDENT PROGRAMS

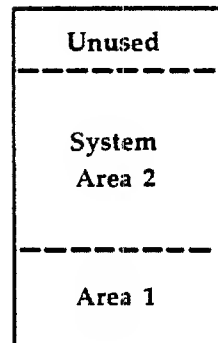
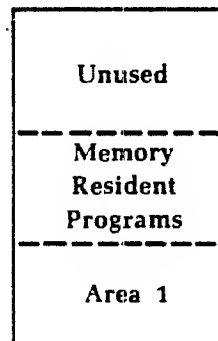
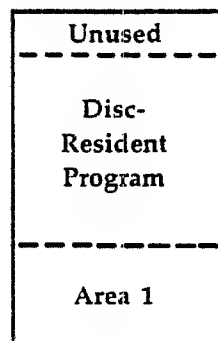
RTE also allows you to define Memory Resident Programs which

- are resident in memory at all times
- are not swappable
- are included in the system when it is generated

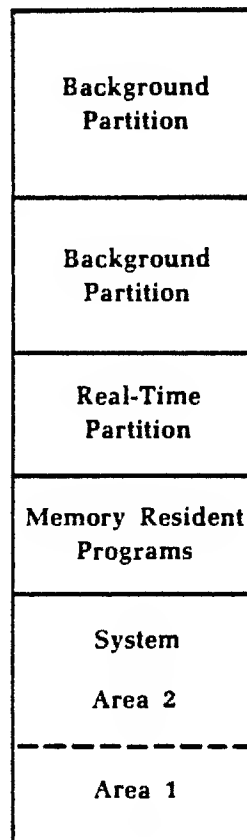


RTE and MEMORY MANAGEMENT

Logical Memory:
how memory
looks to the
currently
executing
program.

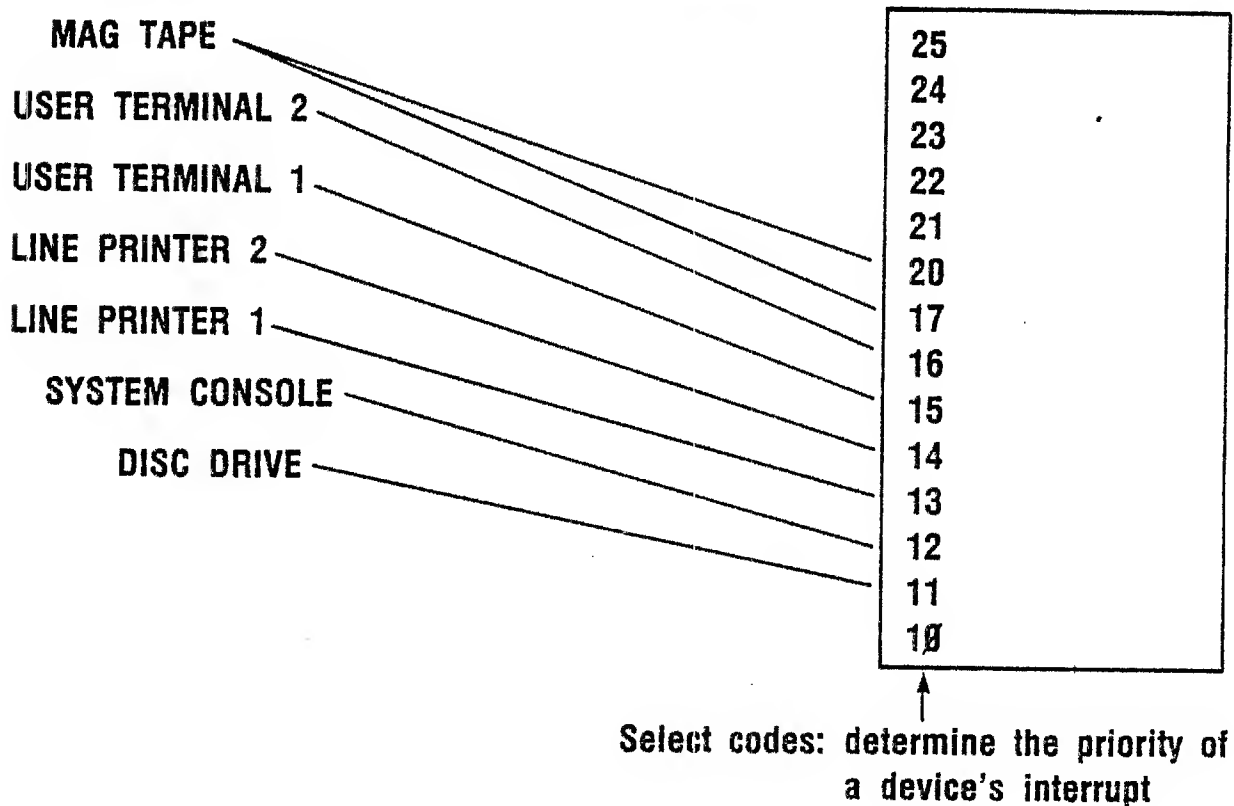


Physical
Memory: the
actual
organization of
RTE in memory.



I/O STRUCTURE

Each peripheral device is connected by a cable to an interface card. The interface card is plugged into a numbered I/O SLOT in the back of the computer.



When an RTE system is generated, the select code assignments are incorporated into RTE's I/O structure.

EQT's

At generation time, each device is assigned an EQUIPMENT TABLE (EQT) number. This number represents an entry in RTE's EQUIPMENT TABLE (EQT).

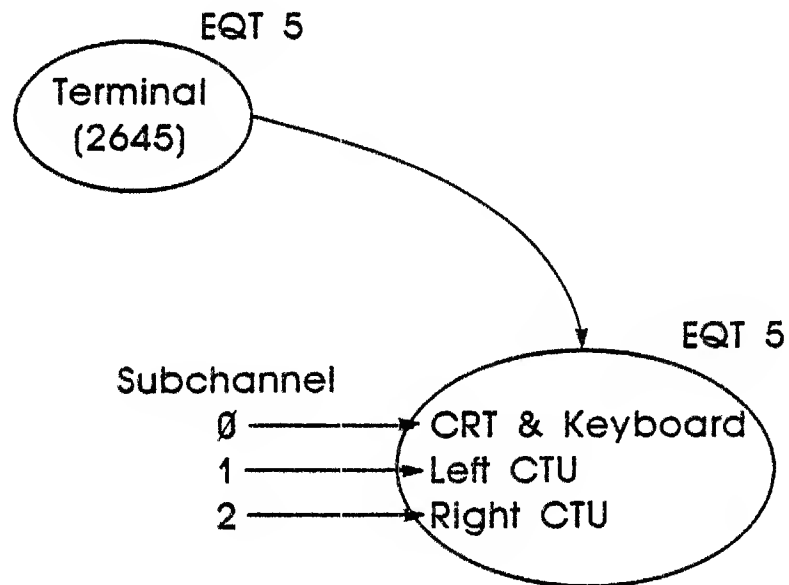
EQT

Eqt numbers		select	driver	last subchannel	
		code		addressed	
1		11	DVR32		(disc)
2		12	DVR05		(system console)
3		13	DVA12		(line printer 1)
4		14	DVA12		(line printer 2)
5		15	DVR05		(user terminal 1)
6		16	DVR05		(user terminal 2)
7		17	DVR23		(mag tape)

SUBCHANNELS

Some devices have several component parts. Each component part is identified by a SUBCHANNEL number.

for example,



SYSTEM LOGICAL UNIT (LU) NUMBERS

When the RTE system is generated, each EQT-SUBCHANNEL pair is assigned a SYSTEM LOGICAL UNIT (LU) number. This represents an entry in RTE's DEVICE REFERENCE TABLE (DRT).

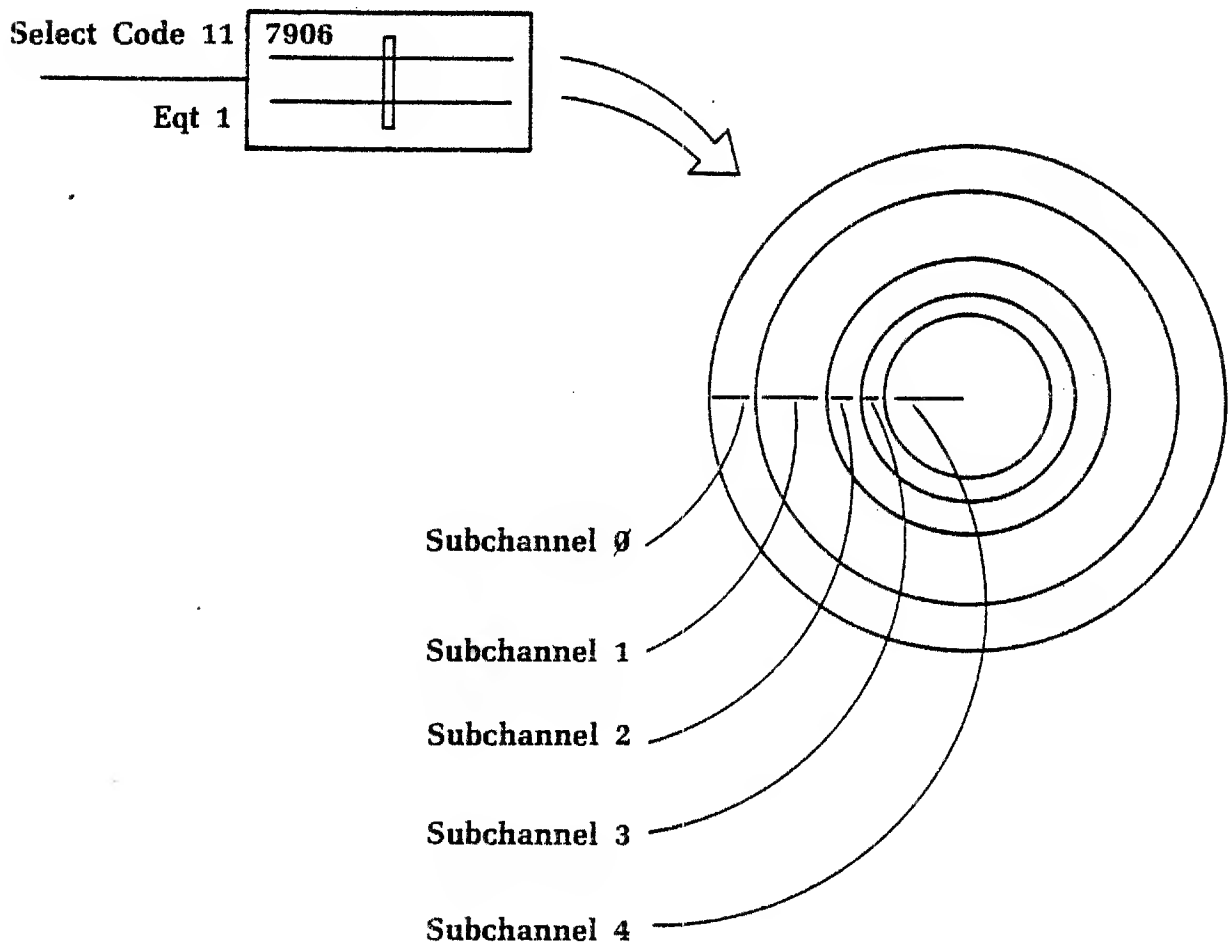
DRT			EQT			
Eqt # Subchannel			select code	driver	last subchannel addressed	
1	2	0	1	11	DVR32	(disc)
2			2	12	DVR05	(system console)
3			3	13	DVA12	(line printer 1)
4	2	1	4	14	DVA12	(line printer 2)
5	2	2	5	15	DVR05	(user terminal 1)
6	3	0	6	16	DVR05	(user terminal 2)
7	4	0	7	17	DVR23	(mag tape)
8	7	0				
...						
...						
30						
31						
32						
...						
...						
65	5	0				
66	6	0				
...						
...						
71	5	1				
72	5	2				
73	6	1				
74	6	2				

Logical Unit (LU) Numbers

DISC CARTRIDGES

When RTE is generated, the System Manager divides the disc into areas, each of which becomes a subchannel.

For example,



DISC LU's

Each disc area (an Eqt-subchannel pair) is assigned a System Logical Unit (LU) number, perhaps:

System LU	Eqt	Subchannel
2	1	0
3	1	4
30	1	1
31	1	2
32	1	3

DRT

Eqt # Subchannel

1	2	0
2	1	0
3	1	4
4	2	1
5	2	2
6	3	0
7	4	0
8	7	0
...		
...		
...		
30	1	1
31	1	2
32	1	3
...		
...		
65	5	0
66	6	0
...		
...		
71	5	1
72	5	2
73	6	1
74	6	2

Logical Unit (LU) Numbers

EQT

select code driver last subchannel addressed

1	11	DVR32		(disc)
2	12	DVR05		(system console)
3	13	DVA12		(line printer 1)
4	14	DVA12		(line printer 2)
5	15	DVR05		(user terminal 1)
6	16	DVR05		(user terminal 2)
7	17	DVR23		(mag tape)



SESSION LU NUMBERS

With SESSION MONITOR, users reference devices via the SESSION LU's set up when their accounts are defined.

For example, if KAREN.PROGDEV logs on at user terminal 1:

DRT

Eqt # Subchannel

1	2	0
2	1	0
3	1	4
4	2	1
5	2	2
6	3	0
7	4	0
8	7	0
:		
:		
30	1	1
31	1	2
32	1	3
:		
:		
65	5	0
66	6	0
:		
:		
71	5	1
72	5	2
73	6	1
74	6	2

Logical Unit (LU) Numbers

SCB

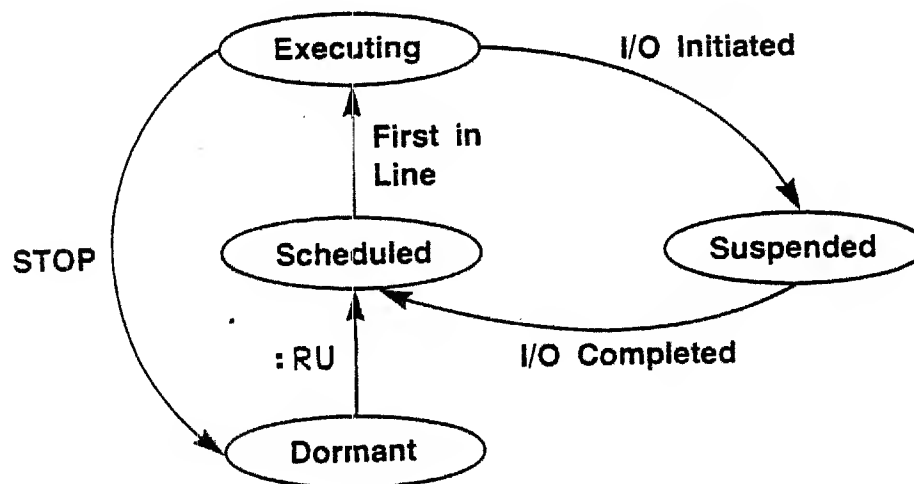
KAREN.PROGDEV	
30	
SST	
System LU	Session LU
65	1
71	4
72	5
2	2
3	3
7	6
8	8

EQT

	select code	driver	last subchannel addressed	
1	11	DVR32		(disc)
2	12	DVR05		(system console)
3	13	DVA12		(line printer 1)
4	14	DVA12		(line printer 2)
5	15	DVR05		(user terminal 1)
6	16	DVR05		(user terminal 2)
7	17	DVR23		(mag tape)

PROGRAM MANAGEMENT

A program's STATE describes the relationship between the program and RTE, which is managing the program's execution.

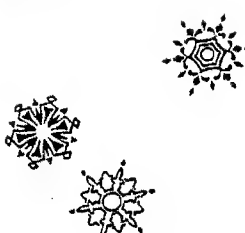


Operator commands, program calls or changes in the environment cause RTE to change a program's state.

Changes in a program's state are called

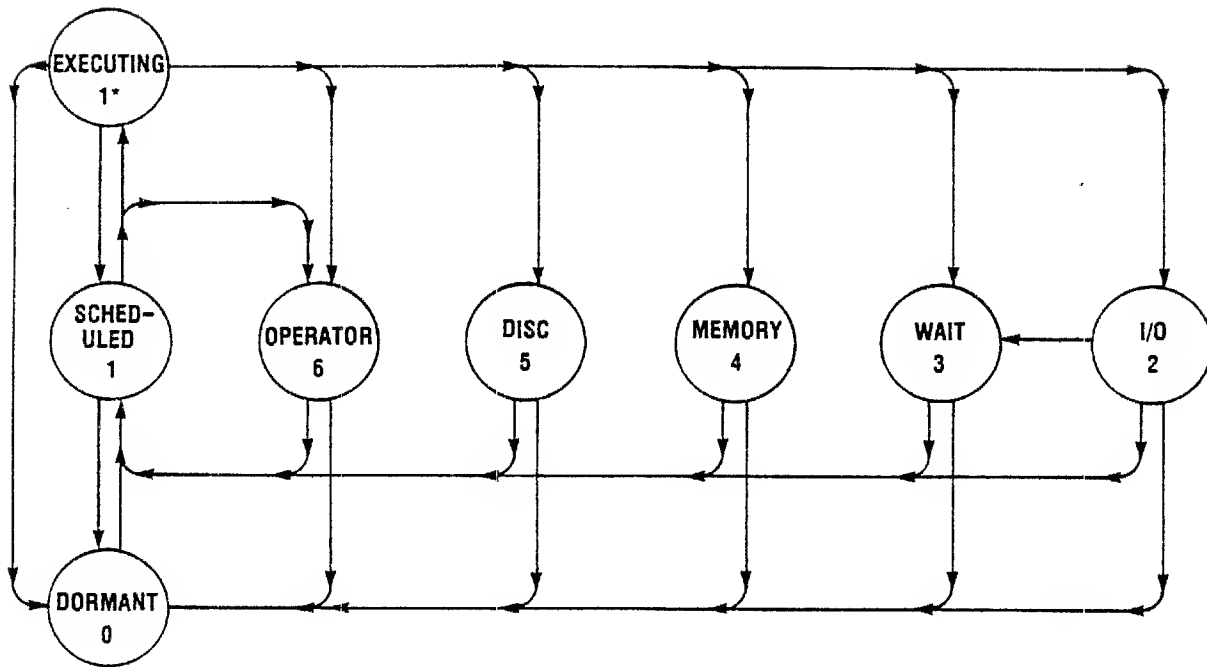
STATE TRANSITIONS

PROGRAM STATES IN RTE



<u>State</u>	<u>Status</u>
0	Dormant
1	Scheduled or Executing
2	I/O suspended
3	General wait
4	Unavailable memory suspend
5	Disc allocation suspend
6	Operator suspend or programmed suspend

USER PROGRAM STATE DIAGRAM



RTE changes a program's state because of operator commands, program requests or changes in the environment.

3C. TROUBLE SHOOTING

Breakmode is usually needed for trouble shooting programs or system operation. Trouble shooting aids include

- breakmode commands

RS

SL

EQ

UP

ST

OF

WH

- utility programs

WHZAT

LGTAT

AN I/O PROBLEM

Suppose you use the FMGR DU command to dump the contents of a file onto a minicartridge in your left CTU.

```
:DU,&PROG,4  
IONR L* 4 E 5 S 1 ***
```

RESTARTING YOUR SESSION

The breakmode RS command aborts and reschedules your Session's copy of FMGR.

```
:DU,&PROG,4  
IONR L* 4 E 5 S 1 ***
```

```
S=65 COMMAND?RS  
FMG65 ABORTED  
:
```


DISPLAYING I/O CONFIGURATION AND STATUS OF DEVICES

The breakmode SL command will

- return the system LU, Eqt number, and status of a specified session LU

S=65 COMMAND ?SL,4
SLU 4=LU #71 = E 5 S 1 D

session LU system LU Eq Subchannel Down

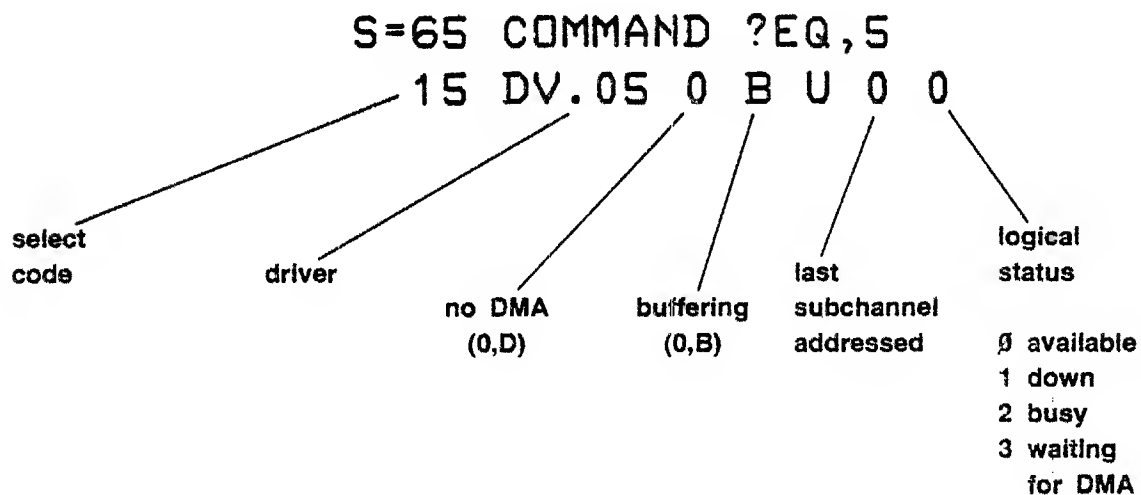
- report all of the user's session LU's if no LU is specified in the command

DISPLAYING EQT INFORMATION

The breakmode EQ command returns information about an EQT entry.

S=xx COMMAND ?EQ,eqt

for example,



MAKING DEVICES AVAILABLE

When a device goes down and the problem is corrected, you must tell RTE that the problem has been fixed.

The breakmode UP command declares an Eqt entry available for I/O.

```
S = xx COMMAND?UP,eqt
```

for example,

```
S =      COMMAND?UP,5
```

BACK TO OUR I/O PROBLEM

```
:DU,&PROG,4  
IONR L* 4 E 5 S 1 ***
```

← You tried to dump a file to your LEFT CTU but got an error message.

```
S=65 COMMAND ?SL,4  
SLU 4=LU # 71 = E 5 S 1 D
```

```
S=65 COMMAND ?EQ,5  
15 DV.05 0 B U 0 0
```

← After having inquired about the LU of the LCTU, you try to “up the Eqt” but get another error message.

```
. S=65 COMMAND ?UP,5  
IONR L* 4 E 5 S 1 ***
```

Looking at the minicartridge, you discover that the “record” slide is set for “write protect.” You slide it to record and —

```
S=65 COMMAND ?SL,4  
SLU 4=LU # 71 = E 5 S 1 D
```

← The device is still down

```
S=65 COMMAND ?UP,5
```

← Tell RTE the device is fixed now

```
:SL,4  
SLU 4=LU # 71 = E 5 S 1  
:
```

← After the dump is complete, we get another FMGR prompt.

SETTING EQT's or DEVICES DOWN



A single device or all devices associated with an EQT may be declared down with the breakmode DN command.

```
S = xx COMMAND?DN,eqt
S = xx COMMAND?DN,,system lu
```

for example,

```
S = 65 COMMAND?DN,4
S = 65 COMMAND?EQ,4
14 DV.12 0 B U 0 1
```

the logical status is



DISPLAY THE STATUS OF A PROGRAM

The breakmode ST command will display a program's status.

- display the status of a specified program

```
S = 21 COMMAND?ST, INF
      99  1  0  0  0  0  0  0
```

priority state deals with time scheduled programs

- display the status of the currently executing program

```
S = 21 COMMAND?ST, 0
R$PN$      1
```

currently executing program its partition

- display the name of the program currently residing in a specified partition

```
S = 21 COMMAND?ST, 4
INF
```

current resident of partition 4

TERMINATING PROGRAMS

The breakmode OF command terminates programs (and possibly releases a program's ID segment and disc tracks).

S = xx COMMAND?OF

will terminate the program most recently run with the FMGR RU command

S = xx COMMAND?OF,program $\begin{bmatrix} ,0 \\ ,1 \\ ,8 \end{bmatrix}$

,0 terminates a program. If the program is I/O suspend, the program is terminated after the I/O operation is completed.

,1 immediately terminates (aborts) a program, clearing any current I/O operation.

,8 immediately terminates (aborts) a program; if the program is temporary on-line loaded, the program is removed from the system.

UTILITY PROGRAM — WHZAT

You can use WHZAT to look at the current status of the programs and partitions in your system.

WHZAT may be run in several ways:

```
                :RU,WHZAT [,lu[,option]]
S = xx COMMAND? RU,WHZAT [,lu[,option]]
```

```
                :WH [,lu[,option]]
S = xx COMMAND? WH [,lu[,option]]
```

where

lu — where to print the display

option — (default)	}	program status displays
AL		
SM		
PA		partition status display

SAMPLE WHZAT PROGRAM STATUS DISPLAY

:WH

14:28:36: 60

PRGRM	T	PRIOR	PT	SZ	DO	SC	IO	WT	ME	DS	OP	.PRG CNTR.	.NEXT TIME.		
**FMG74	3	00090	23	10	*	*	*	*	3	WHZ74	*	*	*	*	P:46363
WHZ74	3	00001	31	4	.	1	P:43177		
TRK	3	00099	34	2	6	.	P:40110		

DOWN LU'S, 97

ALL EQT'S OK

LOCKED LU'S (PROG NAME) 8(JSA01), 11(SPOUT), 70(E..70), 82(E..82),
MAX CONT. FREE TRKS : 73, LU 3

14:28:37:210

:

SAMPLE WHZAT PARTITION STATUS DISPLAY

:WH,,PA

14:28:58:100

PTN#	SIZE	PAGES	BG/RT	PRGRM
1	2	56-	57 BG	QCLM
2	3	58-	60 BG	RQCNV
3	3	61-	63 BG	RPCNV
4	3	64-	66 BG	DLIST
5	4	67-	70 BG	CNSLM
6	8	71-	78 BG	PTOPM
7	8	79-	86 BG	EXECM
8	6	87-	92 BG	EXECW
9	17	93-	109 BG	RFAM
10	17	110-	126 BG	OPERM
11	17	127-	143 BG	PROGL
12	17	144-	160 BG	QUEX
13	17	161-	177 BG	FLUSH
14	17	178-	194 BG	E..70
15	17	195-	211 BG	LOGON
16	17	212-	228 BG	FMG93
17	17	229-	245 BG	R\$PNS
18	17	246-	262 BG	FMG82
19	17	263-	279 BG	SPOUT
20	17	280-	296 BG	SMP
21	17	297-	313 BG	E..82
22	17	314-	330 BG	FMG01
23	17	331-	347 BG	FMG74
24	17	348-	364 BG	LGOFF
25	17	365-	381 BG	FMG68
26	17	382-	398 BG	FMG73
27	17	399-	415 BG	FMG89
28	17	416-	432 BG	FMG94
29	27	433-	459 BG	FIL78
30	27	460-	486 BG	WHZ74
31	27	487-	513 BG	E..94
32	27	514-	540 BG	RUN68
33	27	541-	567 BG	JSA01
34	27	568-	594 BG	TRK
35	17	595-	611 BG	FMG66
36	17	612-	628 BG	E..78
37	11	629-	639 BG	DBONC
38-64	<UNDEFINED>			

14:29: 0:900

:

UTILITY PROGRAM — LGTAT



LGTAT displays information about the tracks on the system disc cartridge (LU 2) and the auxiliary disc cartridge (LU 3).

```
S = xx COMMAND?RU,LGTAT [,lu [,option]]  
      or  
      :RU,LGTAT [,lu [,option]]
```

where

lu — where to display the
information

option — 0 abbreviated output
1 complete output

:RU,LGTAT,1,1
 TRACK ASSIGNMENT TABLE & =PROG ^ =SWAP

TRACK	0	1	2	3	4	5	6	7	8	9
0	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM	AUTOR&	JOB &	,,,,,&	FMGR1&
10	FMGR2&	FMGR4&	FMGR6&	FMGR7&	FMGR9&	LOGON&	LOGON&	LGOFF&	LGOFF&	GASPI&
20	SPOUT&	RFAM &	OPERM&	QCLM &	EDITR&	REMAT&	RQCNV&	HELP &	LOADR&	ACCTS&
30	ACCTS&	ACCT2&	ACCT3&	ACCT4&	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY
40	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY
50	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	LIBRY	ENTS	D.RTR
60	E..68	E..70	WHZAT&	E..78	E..70	E..82	E..66	E..82	TRK &	E..78
70	--	--	E..70	SLXFR&	--	--	MACRO&	PASS1&	POSTP&	POSTP&
80	--	--	--	T5IDM&	GLOBAL	GLOBAL	GLOBAL	GLOBAL	GLOBAL	SFAVG&
90	--	--	--	LSTEN&	TMP &	--	--	--	--	--
100	--	--	--	PRTSV&	PROGA&	MAIN &	--	--	--	--
110	--	--	--	--	--	--	--	--	--	--
120	--	--	--	--	--	--	--	--	--	--
130	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
140	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
150	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
160	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
170	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
180	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
190	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
200	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
210	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
220	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
230	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
240	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
250	FMP	FMP	FMP	FMP	D.RTR					

AUXILIARY DISC

0	--	--	--	--	--	--	--	--	--	--
10	--	--	--	--	--	--	--	--	--	--
20	--	--	--	--	--	--	--	--	--	--
30	--	--	--	--	--	--	--	--	--	--
40	--	--	--	--	--	--	--	--	--	--
50	--	--	--	--	--	--	--	--	--	--
60	--	--	--	--	--	--	--	--	--	--
70	--	--	--	FIL68^	FIL68^	FIL68^	--	--	--	--
80	--	--	FIL82^	FIL82^	FIL82^	FIL94^	FIL94^	FIL94^	FIL70^	FIL70^
90	FIL70^	--	--	--	--	--	--	--	--	--
100	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
110	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
120	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
130	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
140	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
150	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
160	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
170	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
180	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
190	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
200	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
210	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
220	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
230	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
240	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP	FMP
250	FMP	FMP	FMP	FMP	D.RTR					

THE LS TRACK(S) ARE UNDEFINED
 TOTAL AVAILABLE TRACKS = 130
 LARGEST CONTIGUOUS TRACK BLOCK = 73
 :

3D. BREAKMODE vs SYSTEM COMMANDS

To enter a breakmode command from FMGR use

:SYxx,parameters
 ↑
two character breakmode command

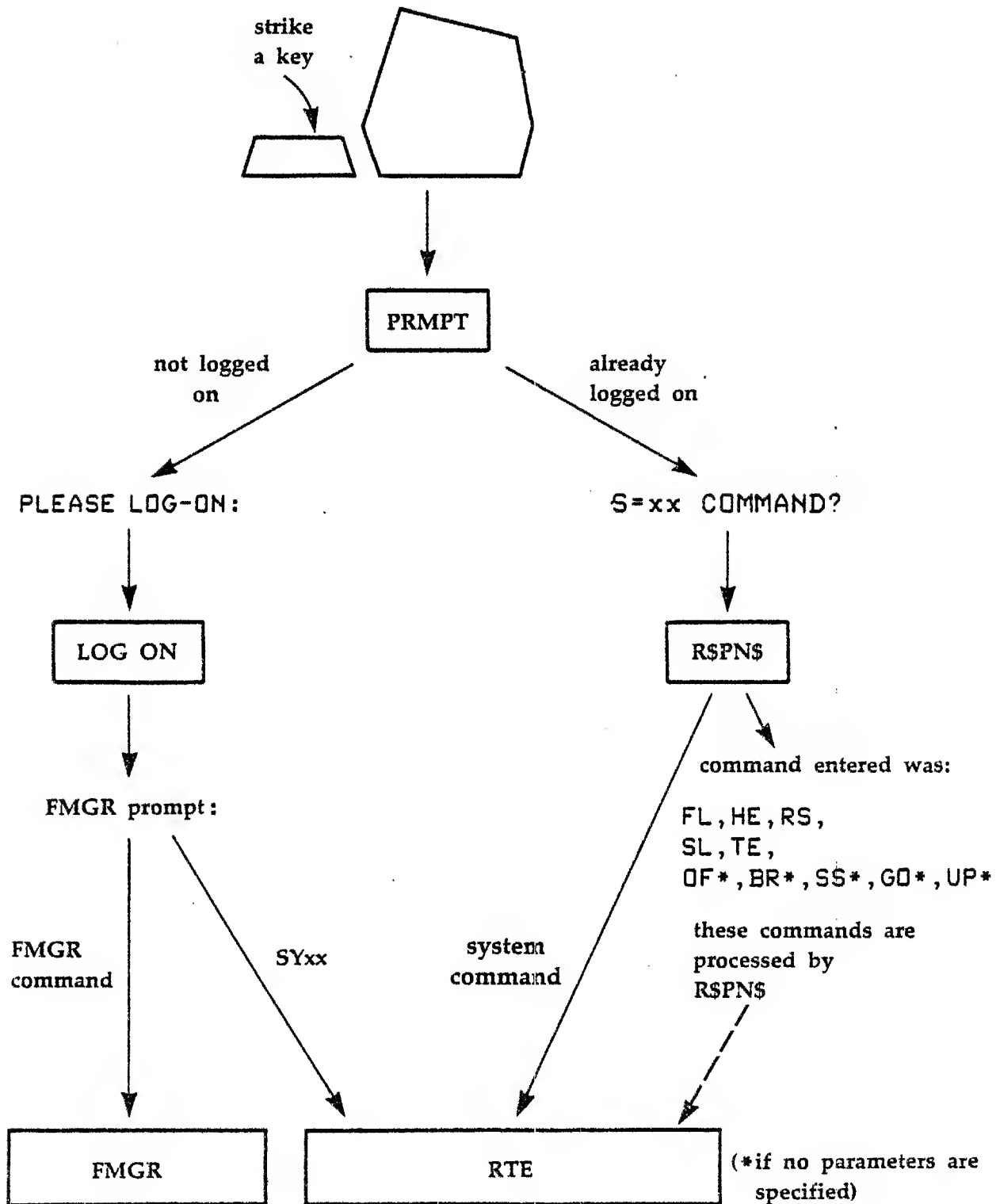
FMGR will pass the breakmode command directly to RTE.

```
:SYTI  
1979 156 15 0 50
```

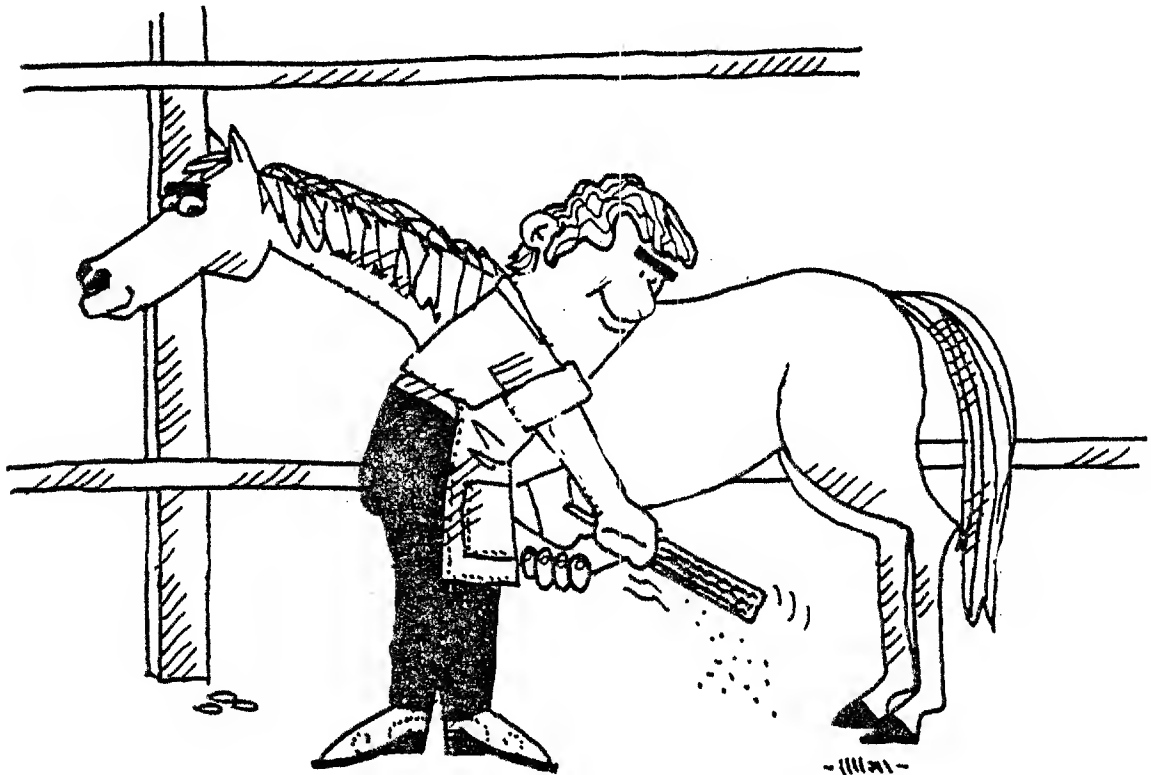
```
:SYEQ,4  
14 DV.12 0 B U 0 0
```

```
:SYOF,PROG6,1
```

BREAKMODE vs SYSTEM COMMANDS



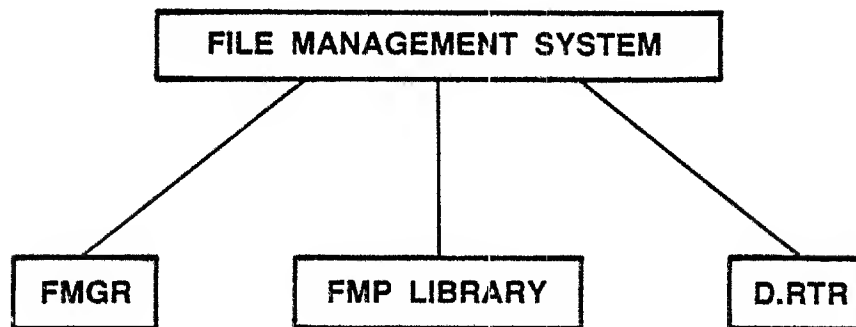
4 FILE MANAGEMENT SYSTEM



SECTION

A	FILE MANAGEMENT SYSTEM OVERVIEW	4-3
B	USING DISC CARTRIDGES	4-8
C	USING FILES	4-19
D	ACCESSING NON-DISC DEVICES	4-28
E	SWITCHING LU'S	4-32

4A. FILE MANAGEMENT SYSTEM OVERVIEW



FMGR

an interactive program that

- interfaces users with RTE
- allows users to manipulate files

FMP LIBRARY a set of routines which manages files

- are used by FMGR
- can be used by your programs

D.RTR

the program which manages file directories

FILE CHARACTERISTICS

Files may be categorized in several ways.

- **METHOD OF ACCESS**

RANDOM vs SEQUENTIAL ACCESS

- **RECORD LENGTH**

FIXED LENGTH vs VARIABLE LENGTH RECORDS

- **FILE EXTENDABILITY**

FIXED LENGTH vs EXTENDABLE FILES

- **FILE CONTENT**

ASCII vs BINARY DATA

FILE TYPES

<u>Category</u>	<u>Type</u>	<u>Description</u>
Non-Disc File	0	Handle non-disc devices just like disc files
Fixed-length record, Random Access, Non-extendable	1	128 word record length
	2	User-defined record length
Variable-length record, sequential access, automatic extents	3	Variable record length
	4	Source program (ASCII)
	5	Relocatable code (binary relocatable)
	6	Memory image program
	7	Absolute Code (binary absolute)
	>7	User defined

*** FILE SIZES

The File Management System allocates space for files in either

- multiples of 128 blocks

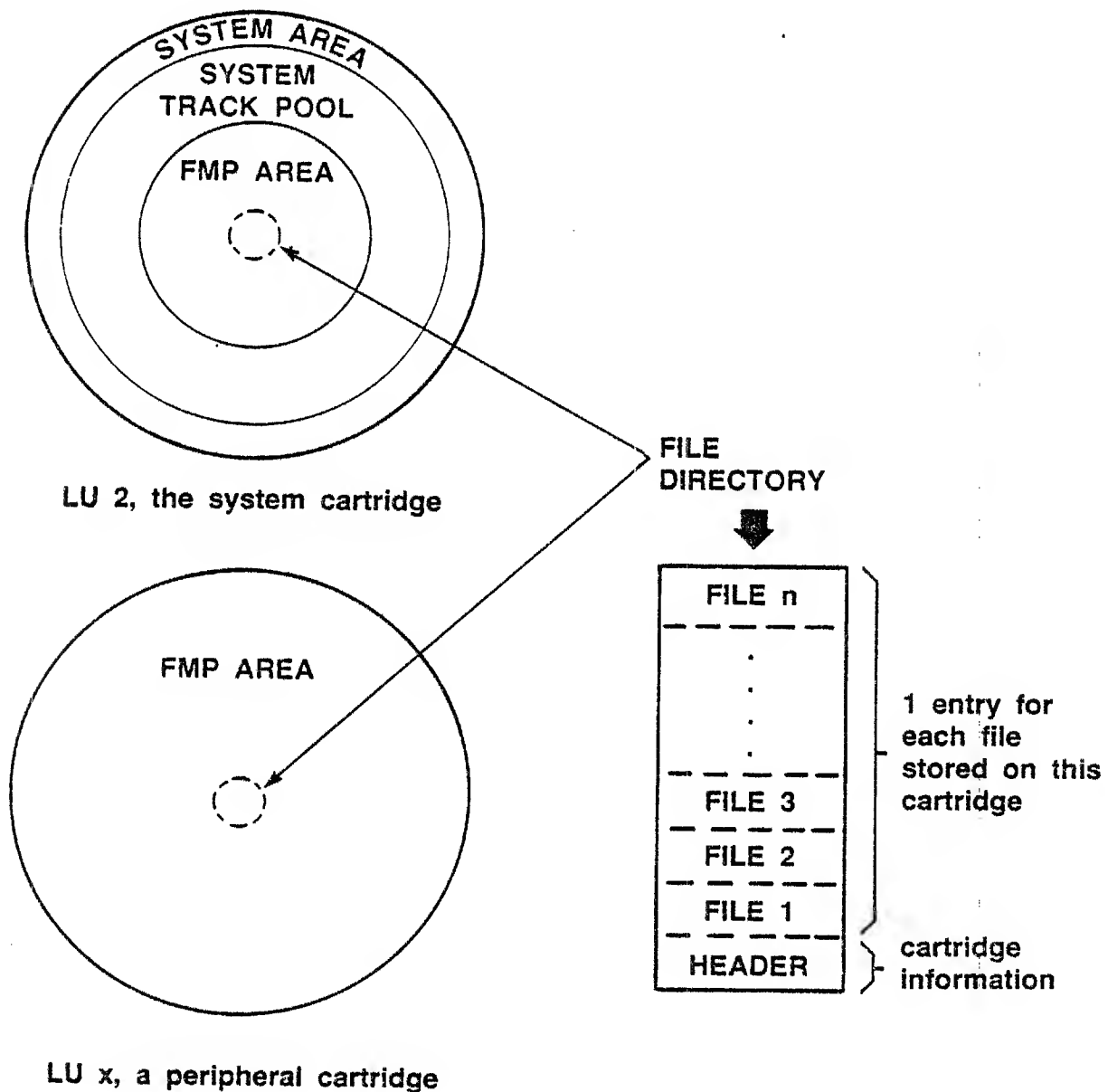
The maximum sizes associated with files are:

	Files allocated by —	
	blocks	128 block multiples
maximum size	16383 blocks	32767 x 128 blocks
maximum record length	32767 words	32767 words
maximum number of records per file	$2^{15}-1$	$2^{31}-1$

(1 block = 128 words)

FILES AND DISC CARTRIDGES

Files may be stored in the user file areas (FMP areas) of LU 2 or LU 3 or on any other peripheral cartridge.



4B. USING DISC CARTRIDGES

- *Disc cartridges (or disc subchannels) are defined and assigned LU numbers when an RTE system is generated.*
- *Before you can use disc files via the File Management System, you must tell the File Management System that you want to use a particular disc cartridge to hold your disc files. This is done by using a FMGR command to “mount a cartridge”.*
- *Session Monitor permits you to use only those disc cartridges which were*
 - mounted by you*
 - not mounted by you but are still available for your use.*

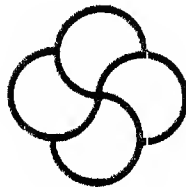
YOUR CARTRIDGE LIST

The FMGR CL command displays a list of those disc cartridges which you can access.

For example, if KAREN.PROGDEV does a CL —

:CL

LU	LAST TRACK	CR	LOCK	P/G/S
32	00140	SS		P
30	00140	01500		G
02	00255	00002		S
03	00255	00003		S
31	00400	00031		S



DISC LOGICAL UNIT NUMBERS VS CARTRIDGE REFERENCE NUMBERS

:CL

LU	LAST TRACK	CR	LOCK	P/G/S
32	00140	SS		P
30	00140	01500		G
02	00255	00002		S
03	00255	00003		S
31	00400	00031		S

Disc cartridges have two identifiers:

- **LOGICAL UNIT (LU) NUMBERS** — are assigned when the disc cartridges are defined during the generation of an RTE system. For disc cartridges, the **SYSTEM LU** and **SESSION LU** numbers are the same.
- **CARTRIDGE REFERENCE NUMBERS (CRN's)** — are alternate identifiers for disc cartridges. They are usually assigned by users when cartridges are mounted.

RESTRICTED CARTRIDGE ACCESS

:CL

LU	LAST TRACK	CR	LOCK	P/G/S
32	00140	SS		P
30	00140	01500		G
02	00255	00002		S
03	00255	00003		S
31	00400	00031		S

The P/G/S indicates which user "mounted" that cartridge and which user(s) can access that cartridge.

For the cartridge list displayed by KAREN.PROGDEV,

 P — PRIVATE

The cartridge was "mounted" by KAREN.PROGDEV and can only be accessed by her.

 G — GROUP

The cartridge was "mounted" by a user in the PROGDEV group and can be accessed by any user in that group.

 S — SYSTEM

These cartridges "belong" to the System Manager but can be accessed by any user (LU 2 and 3 are "read only" however).

DISPLAYING ALL CARTRIDGES

The CL command lists those cartridges which a user can access; suppose PETER.ADMIN enters a CL —

:CL

LU	LAST TRACK	CR	LOCK	P/G/S	
02	00255	00002		S	} these are the cartridges which PETER.ADMIN can access
03	00255	00003		S	
31	00400	00031		S	

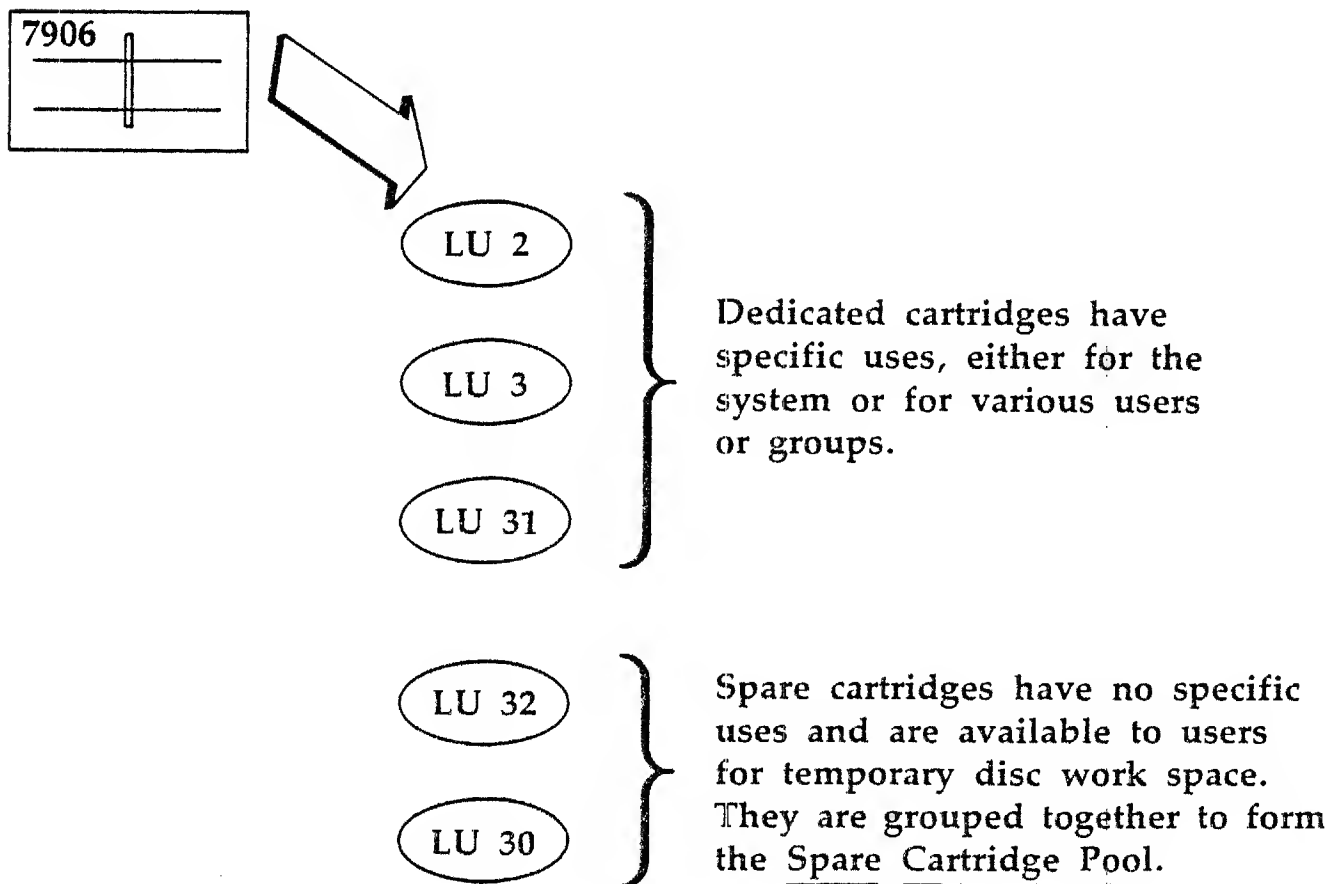
An expanded version of the CL command lists *all* cartridges mounted and indicates who "owns" those cartridges. Suppose PETER.ADMIN enters a CLAL —

:CLAL

LU	LAST TRACK	CR	LOCK	P/G/S	USER/GROUP
02	00255	00002		S	MANAGER.SYS
03	00255	00003		S	MANAGER.SYS
30	00140	01500		G	PROGDEV
31	00400	00031		S	MANAGER.SYS
32	00140	SS		P	KAREN.PROGDEV

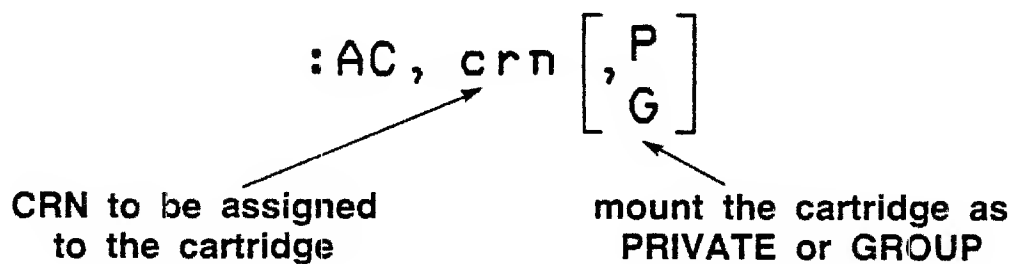
DEDICATED vs SPARE CARTRIDGES

When setting up the Session Monitor Accounts System, the System Manager can classify the disc cartridges defined in the system to be either DEDICATED or SPARE cartridges. For example,



ALLOCATING A CARTRIDGE FROM THE SPARE CARTRIDGE POOL

Use the FMGR AC command to allocate (mount)
a cartridge from the Spare Cartridge Pool



The AC command will:

- find the first available cartridge in the pool
- initialize the cartridge
- mount the cartridge

EXAMPLES OF ALLOCATING CARTRIDGES

*KAREN.PROGDEV might have done the following
to mount cartridges:*

```
:CL
  LU  LAST TRACK   CR   LOCK  P/G/S
  02      00255    00002          S
  03      00255    00003          S
  31      00400    00031          S
```

```
:AC,SS
```

```
:CL
  LU  LAST TRACK   CR   LOCK  P/G/S
  32      00140      SS          P
  02      00255    00002          S
  03      00255    00003          S
  31      00400    00031          S
```

```
:DL,SS
```

```
CR= SS
```

```
ILAB=DC0032 NXTR= 00000 NXSEC=000 #SEC/TR=096 LAST TR=00140 #DR TR=01
```

```
NAME      TYPE      SIZE/LU      OPEN TO
```

```
:AC,1500,G
```

```
:CL
  LU  LAST TRACK   CR   LOCK  P/G/S
  32      00140      SS          P
  30      00140    01500          G
  02      00255    00002          S
  03      00255    00003          S
  31      00400    00031          S
```

```
:DL,1500
```

```
CR=01500
```

```
ILAB=DC0030 NXTR= 00000 NXSEC=000 #SEC/TR=096 LAST TR=00140 #DR TR=01
```

```
NAME      TYPE      SIZE/LU      OPEN TO
```

```
:AC,1000
```

```
FMGR 064
```

```
:??
```

```
FMGR 064 NO DISCS AVAILABLE FROM DISC POOL
```

```
:
```

YOUR SCB CARTRIDGE LIST



- *Your SCB contains a list of the PRIVATE or GROUP cartridges currently mounted to your account or to your group.*

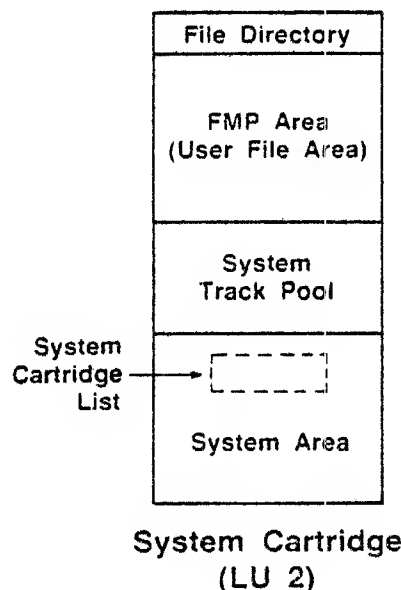
SCB	
KAREN.PROGDEV	
30	
SST	
system LU	session LU
65	1
71	4
72	5
2	2
3	3
7	6
8	8
31	31
32	32
30	30
cartridge list:	32, 30

LU	LAST TRACK	CR	LOCK	P/G/S
32	00140	SS		P
30	00140	01500		G
02	00255	00002		S
03	00255	00003		S
31	00400	00031		S

SYSTEM CARTRIDGE LIST VS YOUR SCB'S CARTRIDGE LIST

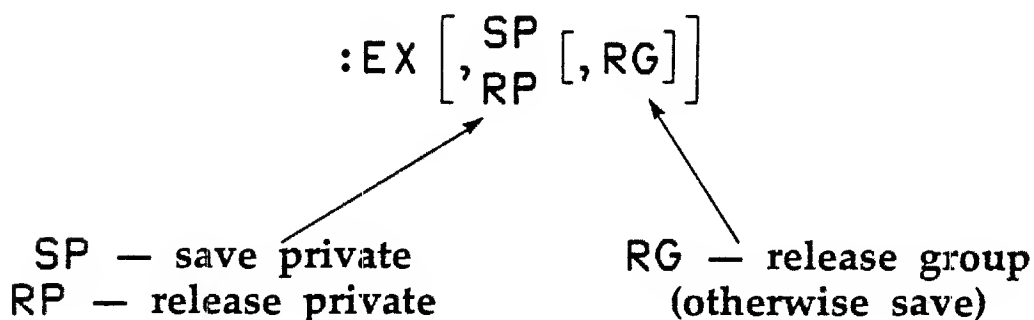
In addition to the cartridge list in your SCB, the File Management System maintains a list of all currently mounted cartridges.

This list, the **SYSTEM CARTRIDGE LIST**, is kept in the system area of the system cartridge (LU 2).



LOG OFF

Parameters to the FMGR EX command specify whether to save or return cartridges when you log off.



When you log-on again, any private or group cartridges saved are still available for your use.

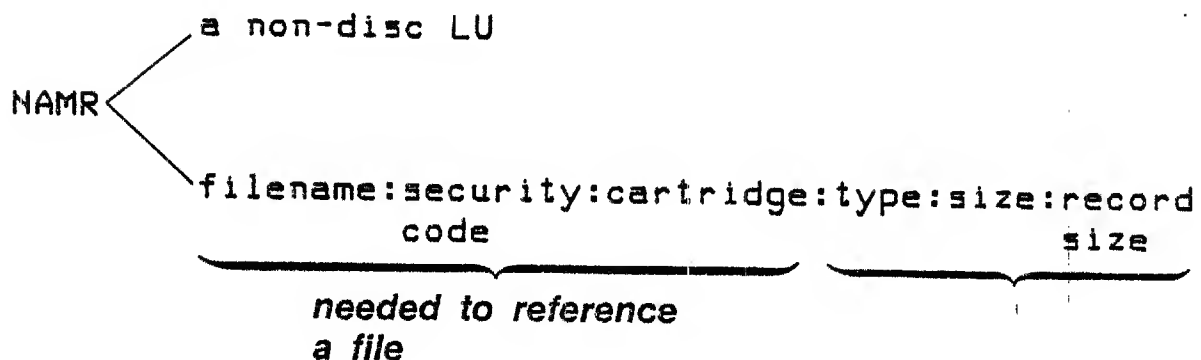
4C. USING FILES

Many FMGR commands can refer to either disc files or non-disc devices, for example.

:LI, file name
lu

↑
the parameter can be either the name of a disc file or the LU of a non-disc device

The term NAMR refers to such a parameter:



WHY SPECIFY CARTRIDGES?



Suppose you have two files, both called "DATA".

LU	LAST TRACK	CR	LOCK	P/G/S	
32	00140	SS		P	→ first "DATA" stored here
30	00140	01500		G	
02	00255	00002		S	
03	00255	00003		S	→ second "DATA" stored here
31	00400	00031		S	

When you run EDITR,

```
:RU,EDITR  
SOURCE FILE?  
/DATA
```

which file will you
be editing?

SEARCHING THE FILE DIRECTORY

- Use the FMGR DL command to list the file directory of a specified cartridge.

:DL[,cartridge]

↑
positive CRN or
negative LU

- Use the FMGR DL command to search for a particular file on one or all of the cartridges in your cartridge list.

:DL,namr

↑
if a cartridge is specified, search only
that cartridge, otherwise search all the
cartridges in your cartridge list

- For example,

```
:DL,DATA
:DL,SOURCE:::3
:DL,&-----::-24
```

CREATING FILES -

ASCII source, text or data files may be created by using EDITR.

```
:RU,EDITR
SOURCE FILE?
/O
EOF
/ΔFTN4,L
/Δ;PROGRAM MAIN
/Δ;INTEGER PARM(5)
/Δ .
.
.
.
.
/Δ;END
/Δ;END$
/EC filename:security code:cartridge
```

 file to be created

EDITR will create a type 4 file whose size is the number of blocks needed to contain the file being created.

(Δ = space)

(; = EDITR tab character)

CREATING FILES —



You can also create a file and store data in it with the FMGR ST command.

`:ST,namr1,namr2[,record format]`

a non-disc LU or disc file containing data to be stored in the new file	file <u>to be created</u> and <u>filled with</u> data, or a non-disc LU to receive the data	type of data in namr1: BR — binary relocatable AS — ASCII
--	---	--

If a file is specified for namr2, it must not already exist; the ST command tries to create a file for you.

for example,



use the ST command to enter a file from your terminal:

```
:ST,1,DATA:FF  
JACK  
SAM  
LINDA  
PETER  
MARY
```

Ⓓ

:

(Control — D acts as an EOF to terminate input)



input a binary relocatable file from a mag tape (LU 8):

```
:ST,8,%PROG,BR
```



duplicate a file on another cartridge:

```
:ST,TSPD::DP,TSPD::2000
```

CREATING FILES —

To create a file without supplying data, use the FMGR CR command:

:CR, namr

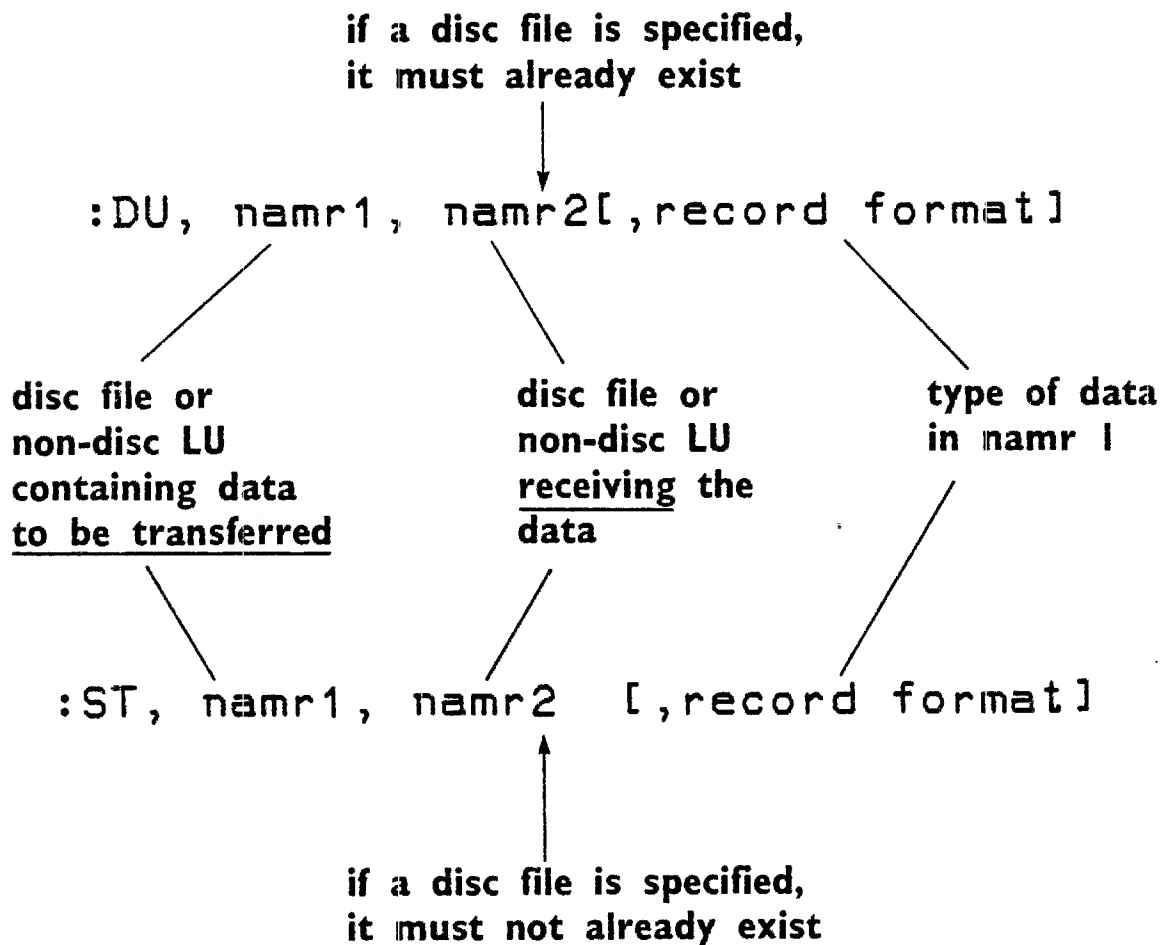
↑
file to be created; in addition to the file name you must specify file type, file size (and record length for type 2 files).

for example, if you wish to list the file directories of all your cartridges but store the list in a file, you might:

```
:CR, DLIST::-24:4:10  
:LL, DLIST  
:DL  
:LL, 1
```

STORING vs DUMPING FILES

Both the FMGR ST and DU commands allow data to be transferred between a source namr and a destination namr.



MORE FILE MANIPULATIONS

PURGING FILES

```
:PU, namr
```

file to be purged

RENAMING FILES

```
:RN, namr, new file name
```

file to be renamed

its new name

LISTING FILES

```
:LI, namr[,format[,line 1[,line 2]]]
```

lines to be listed

list in:

S ASCII format

B binary format

D directory info only

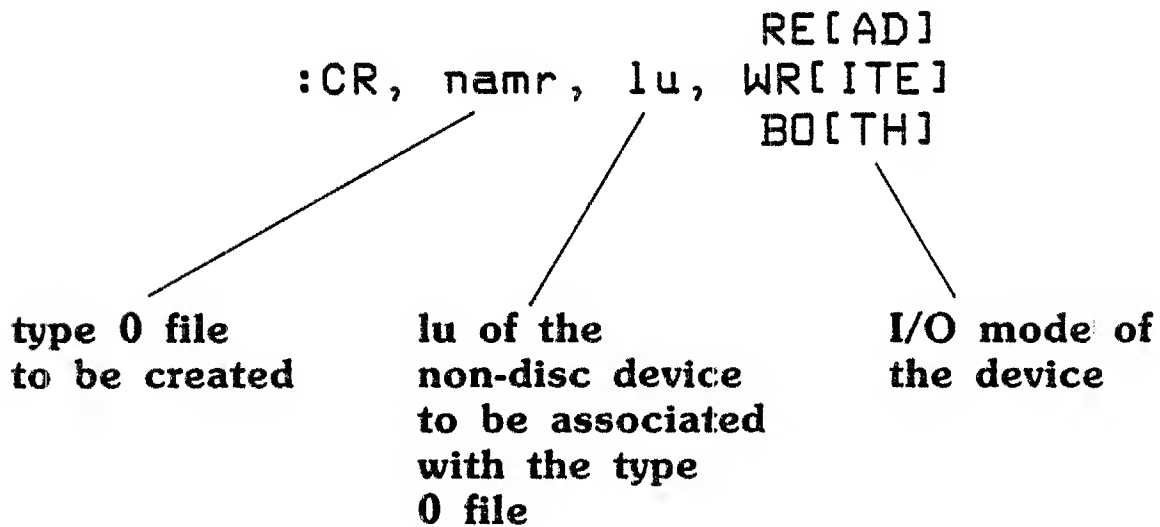
4D. ACCESSING NON-DISC DEVICES

You can also use FMGR commands to manipulate non-disc devices, either by referring to their LU numbers or to type 0 files associated with the devices.

TYPE 0 FILES



Type 0 files allow you to refer to a non-disc device by a file name rather than by the device's LU number.



CONTROLLING A NON-DISC DEVICE

The CN command allows you to control devices via their LU numbers or type 0 files.

`:CN [,namr [,function [,subfunction]]]`

The diagram shows the command structure with three labels and lines pointing to specific parts of the command:

- device to be controlled** points to the `namr` parameter.
- function to be performed** points to the `function` parameter.
- more control information** points to the `subfunction` parameter.

☆☆☆☆☆☆☆☆

CONTROLLING TERMINALS

Terminals may be controlled with the FMGR CN or CT commands.

`:CT ,namr[,function[,subfunction[,message]]]`

interactive
device

control function
to be performed

additional control
information

message to be
displayed on
the terminal

4E. SWITCHING LU'S

In addition to displaying the LU's in your SST, the SL command may be used to

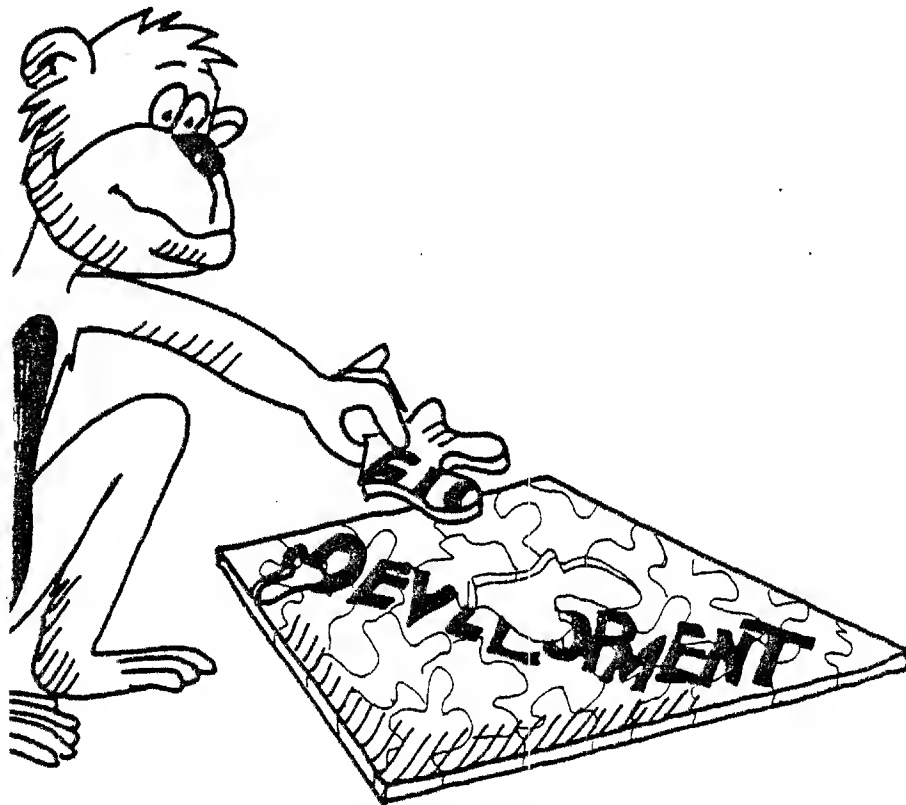
- add a new Session LU

```
:SL,4
SLU 4=LU #120 = E19 S 1
:SL,10,120
:SL,10
SLU 10=LU #120 = E19 S 1
:SL,10,-
:SL,10
SLU 10=NOT DEFINE
:
```

- add a new System LU

```
:SL,6
SLU 6=LU # 6 = E 6
:SL,6,7
:SL,6
SLU 6=LU # 7 = E 7
:SL,6,-
:SL,6
SLU 6=LU # 6 = E 6
:
```

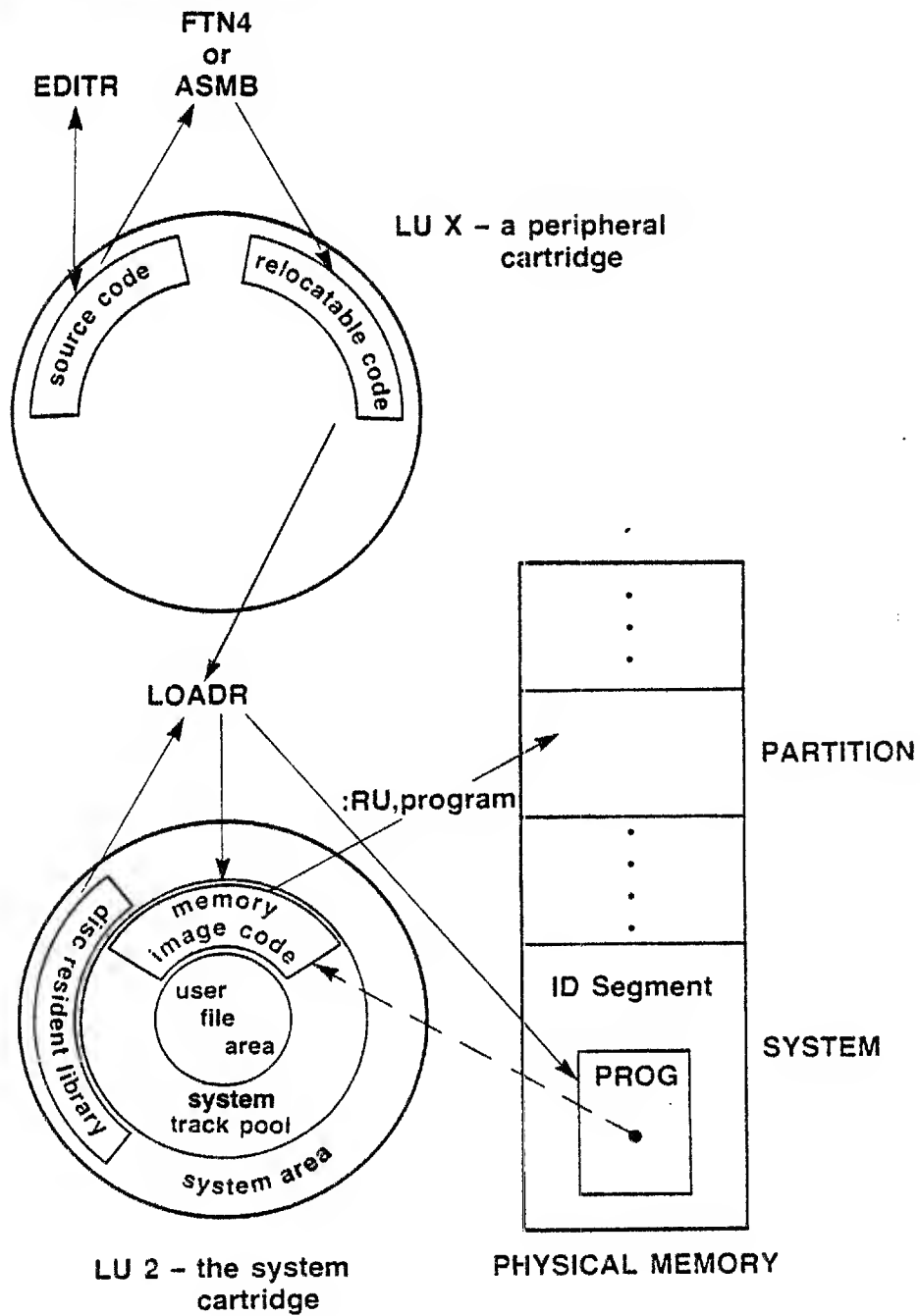
5 PROGRAM DEVELOPMENT



SECTION

- A THE PROGRAM DEVELOPMENT PROCESS**
- B FTN4 AND ASMB**
- C USING LOADR**
- D COMPL/CLOAD**

5A. THE PROGRAM DEVELOPMENT PROCESS



5B. FTN4 AND ASMB

FORTTRAN source programs have this format:

FTN4, compiler options

```
PROGRAM name [(type,priority,res,mult,hr,min,sec,msec),comments]  
.  
.  
.  
.  
.  
.  
.  
.  
.  
.  
END  
END$
```

ASSEMBLER source programs have this format:

ASMB, assembler options

```
NAM name [,type,priority,res,mult,hr,min,sec,msec & id]  
.  
.  
.  
.  
.  
.  
.  
.  
END entry
```



INVOKING FTN4 OR ASMB

You run FTN4 or ASMB by using the FMGR or system RU command.

```
:RU, FTN4, source, list, binary, line count, options  
ASMB
```

- source** — FILE OR LU containing the source code
- list** — file or LU to receive the listing of the compilation or assembly
- binary** — file or LU to receive the relocatable code
- line count** — number of lines to be printed per page
- options** — any compiler/assembler options specified here replace those specified in the control statement.



FILE NAMING CONVENTIONS FOR PROGRAM DEVELOPMENT

First character of file name	Type of file
&	Source code file
'	List file
%	Relocatable code file



If the source file name begins with "&", then the compiler or assembler will use a list or binary file whose name begins with "/" or "%" respectively.

For example, :RU,FTN4,&PROG,1,—

the compiler will store the relocatable code
in file "%PROG"

:RU, ASMB, &APGM, -, -

the assembler will store the listing in file
 "APGM" and the relocatable code in file
 "%APGM"

The list and binary files will:

- reside on the same cartridge as the source file
- will be given the security code of the source file
- will be created or replaced as needed



INTERFACING FTN4 AND ASMB

FORTRAN programs and subprograms can invoke
Assembler subprograms

OR

Assembler programs and subprograms can invoke
FORTRAN subprograms

IF

you follow the “.ENTR calling sequence.”

.ENTR is a subprogram (in RTE's relocatable library) designed to handle passing parameters between programs and subprograms or subprograms and other subprograms.

FTN4 uses .ENTR for your subprograms in FORTRAN. If you code your Assembler programs or subprograms to use .ENTR, then they will be compatible with FORTRAN programs or subprograms.



5C. USING LOADR

After you have compiled (or assembled) a source program, you need to use LOADR to *relocate* and *link* the resulting relocatable code.

```
0001  FTN4,L,Q
0002  00000      PROGRAM MAIN
0003  00000      DIMENSION DATA(10)
0004  00000 C
0005  00000 C      INPUT 10 VALUES
0006  00000 C
0007  00000      DO 20 I = 1,10
0008  00031      WRITE(1,101) I
0009  00041 101  FORMAT(' INPUT VALUE ',I2,' ?_')
0010  00041 20    READ(1,*) DATA(I)
0011  00041 C
0012  00041 C      USE SUBROUTINE STAT TO FIND AVERAGE, STANDARD DEVIATION
0013  00041 C
0014  00063      CALL STAT(DATA,10,AVG,STDDEV)
0015  00063 C
0016  00063 C      OUTPUT RESULTS
0017  00063 C
0018  00071      WRITE(1,102) AVG, STDDEV
0019  00103 102  FORMAT('RESULTS ARE:'/
0020  00103      1      'AVERAGE - ',F10.5/
0021  00103      2      'STANDARD DEVIATION - ',F10.5)
0022  00103 C
0023  00103      END

0024  00175      SUBROUTINE STAT(ARRAY,NELE,AVG,STDDEV)
0025  00175 C
0026  00000      DIMENSION ARRAY(NELE)
0027  00000 C
0028  00000      SUM = 0.0
0029  00015      SUMSQ = 0.0
0030  00015 C
0031  00015 C      FIND AVERAGE OF ARRAY ELEMENTS
0032  00015 C
0033  00021      DO 50 I = 1,NELE
0034  00023 50    SUM = SUM + ARRAY(I)
0035  00044      AVG = SUM / FLOAT(NELE)
0036  00044 C
0037  00044 C      FIND STANDARD DEVIATION
0038  00044 C
0039  00056      DO 60 I = 1,NELE
0040  00060      DEV = ARRAY(I) - AVG
0041  00072 60    SUMSQ = SUMSQ + DEV**2
0042  00111      STDDEV = SQRT(SUMSQ)
0043  00111 C
0044  00117      RETURN
0045  00120      END
```

PROGRAM RELOCATION

LOADR displays a "load map" showing the results of relocating and linking the program in the specified relocatable code file.

First, the modules in the relocatable file are "loaded"

```
:RU,LOADR,,%MAIN,1
  MAIN  40042 40236
  STAT  40237 40377
```

FMTIO	40400	41676	24998-16002	REV.1926	790417
FMT.E.	41677	41677	24998-16002	REV.1901	781107
PNAME	41700	41745	771121	24998-16001	
REIO	41746	42072	92067-16268	REV.1903	790316
ERRO	42073	42162	771122	24998-16001	
SQRT	42163	42264	780424	24998-16001	
.RTOI	42265	42360	780921	24998-16001	
.FPWR	42361	42422	781106	24998-16001	
ERO.E	42423	42423	750701	24998-16001	
FRMTR	42424	46061	24998-16002	REV.1926	790503
.CFER	46062	46137	750701	24998-16001	

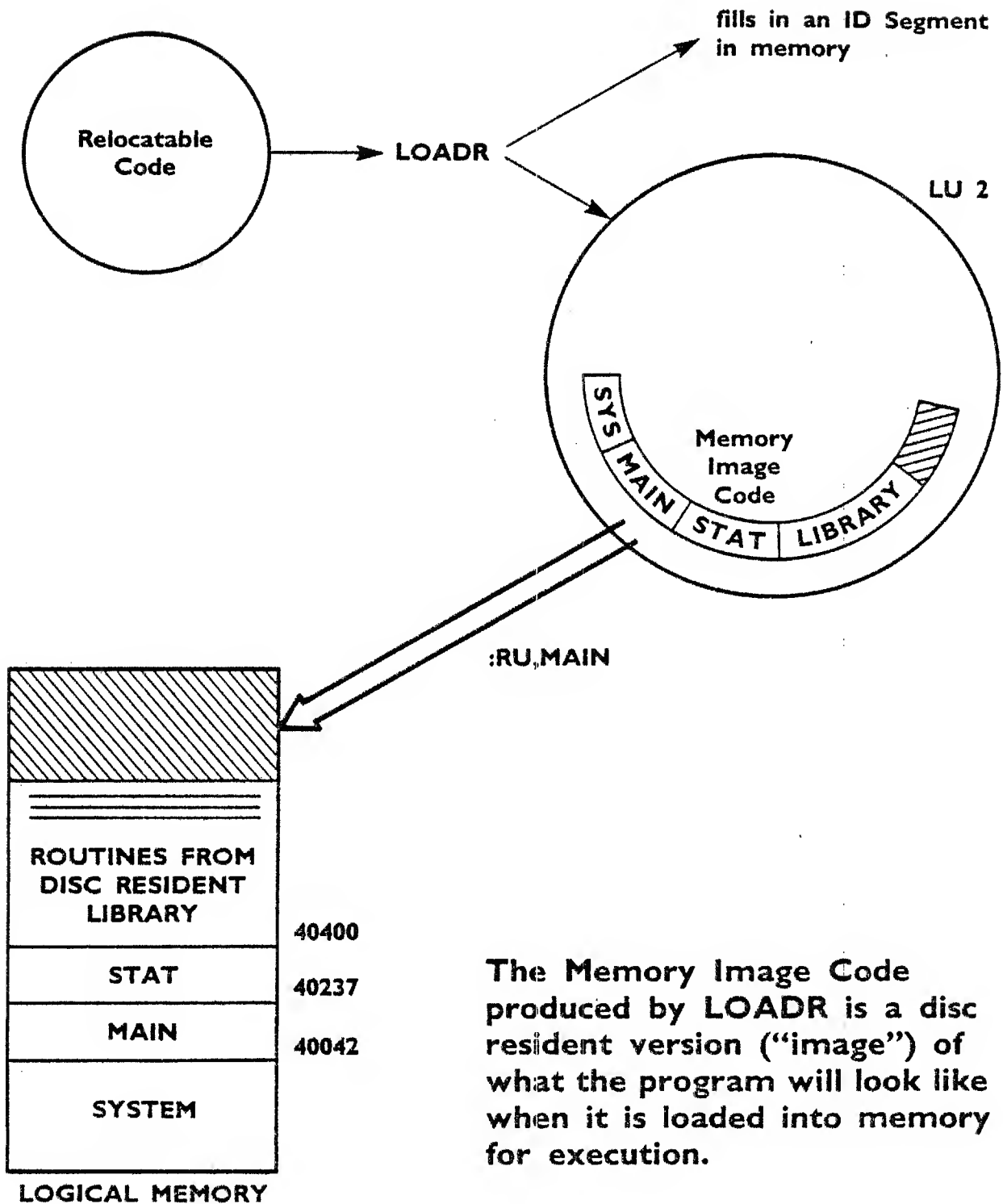
Then, the disc resident library is searched for any routines needed by the program.

```
5 PAGES RELOCATED      5 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP      PROGRAM:BG      LOAD:TE      COMMON:NC
/LOADR:MAIN    READY AT 1:22 PM THU., 1 MAY , 1980

/LOADR:$END
```

Finally, LOADR tells you the name of the program just "loaded."

LOADING PROGRAMS



USING LOADR INTERACTIVELY

If you invoke **LOADR** without any parameters, **LOADR** will prompt you for a command telling it what to do.

```
:RU,LOADR
/LOADR:  RELOCATE,%MAIN
MAIN    40042 40236
STAT    40237 40377
/LOADR:  END

FMTIO    40400 41676    24998-16002 REV.1926 790417
FMT.E    41677 41677    24998-16002 REV.1901 781107
PNAME    41700 41745    771121    24998-16001
REIO     41746 42072    92067-16268 REV.1903 790316
ERRO     42073 42162    771122    24998-16001
SQRT     42163 42264    780424    24998-16001
.RTOI    42265 42360    780921    24998-16001
.FPWR    42361 42422    781106    24998-16001
ERO.E    42423 42423    750701    24998-16001
FRMTR    42424 46061    24998-16002 REV.1926 790503
.CFER    46062 46137    750701    24998-16001

5 PAGES RELOCATED      5 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP      PROGRAM:BG      LOAD:TE      COMMON:NC
/LOADR:MAIN    READY AT 1:23 PM THU., 1 MAY , 1980

/LOADR:$END
```

The **RELOCATE** command says to relocate and link the relocatable code in the specified file.

The **END** command says this is the last command, search the disc resident library to satisfy any remaining unsatisfied external references and create the program.



SEPARATE COMPILATIONS

If relocatable code exists in more than one file, additional RELOCATE commands are used.

```
:RU,LOADR
/LOADR:  RE,%SFSRT
SFSRT  40042 40265    PROGRAM TO INPUT AND SORT INTEGERS
/LOADR:  RE,%BSORT
BSORT  40266 40416    BUBBLE SORT ROUTINE
/LOADR:  EN
```

```
FMTIO  40417 41715    24998-16002 REV.1926 790417
FMT.E  41716 41716    24998-16002 REV.1901 781107
PNAME  41717 41764    771121 24998-16001
REIO   41765 42111    92067-16268 REV.1903 790316
FRMTR  42112 45547    24998-16002 REV.1926 790503
.CFER  45550 45625    750701 24998-16001
```

```
4 PAGES RELOCATED      4 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP  PROGRAM:BG  LOAD:TE  COMMON:NC
/LOADR:SFSRT  READY AT 1:37 PM  THU., 1 MAY , 1980

/LOADR:$END
```



..... DISPLAYING UNDEFS

The **LOADR DISPLAY** command lists all currently unsatisfied or undefined external references (UNDEF's).

```
:RU,LOADR
/LOADR:  RE,%SFSRT
SFSRT  40042 40265    PROGRAM TO INPUT AND SORT INTEGERS
/LOADR:  DI
/LOADR:UNDEFINED EXTS
/LOADR:.DIO.
/LOADR:.IIIO.
/LOADR:.DTA.
/LOADR:EXEC
/LOADR:CLRIO
/LOADR:BSORT
/LOADR:  RE,%BSORT
BSORT  40266 40416    BUEBLE SORT ROUTINE
/LOADR:  EN

FMTIO  40417 41715    24998-16002 REV.1926 790417
FMT.E  41716 41716    24998-16002 REV.1901 781107
PNAME  41717 41764    771121 24998-16001
REIO   41765 42111    92067-16268 REV.1903 790316
FRMTR  42112 45547    24998-16002 REV.1926 790503
.CFER  45550 45625    750701 24998-16001

  4 PAGES RELOCATED      4 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP      PROGRAM:BG  LOAD:TE      COMMON:NC
/LOADR:SFSRT  READY AT  1:39 PM  THU.,  1  MAY , 1980

/LOADR:$END
```



LIBRARIES

- A LIBRARY is a collection of routines which may be used by many different programs.
- The DISC RESIDENT LIBRARY is created when RTE is generated and consists of:
 - the system library
 - the relocatable library
 - user libraries (if included in the generation process)
- LOADR automatically searches the Disc Resident Library when you specify the END command.
- USER LIBRARIES not included in the RTE system must be searched by using the LOADR SEARCH command.



AN EXAMPLE OF A USER LIBRARY

Suppose %SRTL B contains a collection of sorting routines —

%SRTL B →

BSORT (bubble sort)
HSORT (heap sort)
QSORT (quick sort)
SSORT (shell sort)
.
.
.

:RU,LOADR

/LOADR: RE,%PROGA

PROGA 40042 40265

SPECTRUM DATA ANALYSIS

/LOADR: SE,%SRTL B

BSORT 40266 40416

BUBBLE SORT ROUTINE

/LOADR: EN

FMTIO 40417 41715 24998-16002 REV.1926 790417

FMT.E 41716 41716 24998-16002 REV.1901 781107

PNAME 41717 41764 771121 24998-16001

REIO 41765 42111 92067-16268 REV.1903 790316

FRMTR 42112 45547 24998-16002 REV.1926 790503

.CFER 45550 45625 750701 24998-16001

4 PAGES RELOCATED 4 PAGES REQ'D NO PAGES EMA

NO PAGES MSEG

LINKS:BP PROGRAM:BG LOAD:TE COMMON:NC

/LOADR:PROGA READY AT 1:41 PM THU., 1 MAY, 1980

/LOADR:SEND

REAL-TIME VS BACKGROUND PROGRAMS

When you load a program, you can specify it to be a

REAL-TIME DISC RESIDENT PROGRAM

or a

BACKGROUND DISC RESIDENT PROGRAM

The **LOADR** command —

OP,RT specifies **REAL-TIME**

OP,BG specifies **BACKGROUND**

LOCAL COMMON

The following main program and subprogram share a LOCAL (unnamed or blank) COMMON BLOCK.

```
FTN4,L
      PROGRAM MAIN
      COMMON//IAR(500)
      .
      .
      .
      END
      SUBROUTINE SUBR
      COMMON//IAR(500)
      .
      .
      .
      END
```

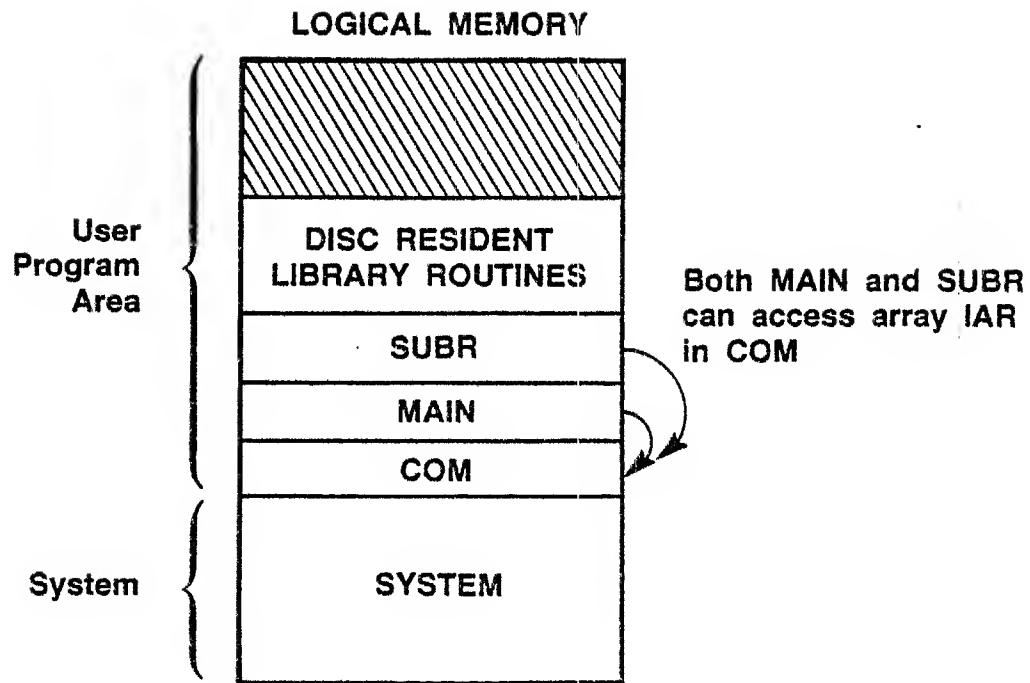
:RU,LOADR

```
/LOADR:  RE,%MAIN
COM      40042 41025
MAIN     41026 41077
SUBR     41100 41117
/LOADR:  EN
```

```
FMTIO  41120 42416  24998-16002 REV.1926 790417
FMT.E   42417 42417  24998-16002 REV.1901 781107
PNAME   42420 42465  771121  24998-16001*
REIO    42466 42612  92067-16268 REV.1903 790316
FRMTR   42613 46250  24998-16002 REV.1926 790503
.CFER   46251 46326  750701  24998-16001
```

```
5 PAGES RELOCATED      5 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP      PROGRAM:BG  LOAD:TE  COMMON:NC
/LOADR:MAIN    READY AT  1:45 PM  THU.,  1  MAY , 1980
```

/LOADR:\$END

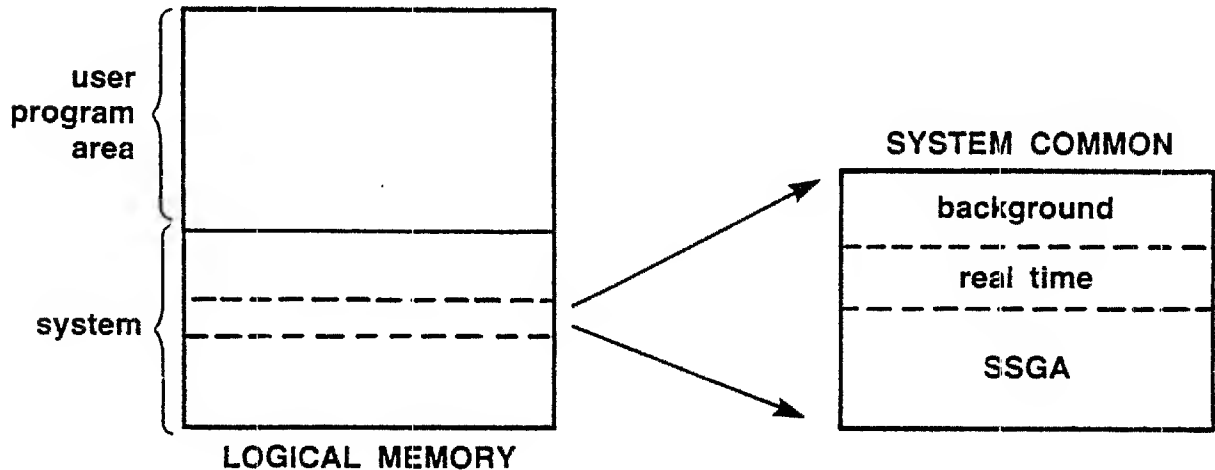


A LOCAL COMMON BLOCK

- **can be shared between a main program and its subprograms or just between subprograms**
- **is within the program itself**
- **is swapped with the program**

SYSTEM COMMON

An UNLABELED COMMON BLOCK may be shared between programs if the COMMON BLOCK is associated with SYSTEM COMMON.



The LOADR command $\square P, \begin{matrix} SC \\ RC \\ NC \\ SS \end{matrix}$ allows your program

to access one or more of these System Common areas.

:RU,LOADR

/LOADR: OP,SC

/LOADR: RE,%MAIN

MAIN 40042 40113

SUBR 40114 40133

/LOADR: EN

FMTIO 40134 41432 24998-16002 REV.1926 790417

FMT.E 41433 41433 24998-16002 REV.1901 781107

PNAME 41434 41501 771121 24998-16001

REIO 41502 41626 92067-16268 REV.1903 790316

FRMTR 41627 45264 24998-16002 REV.1926 790503

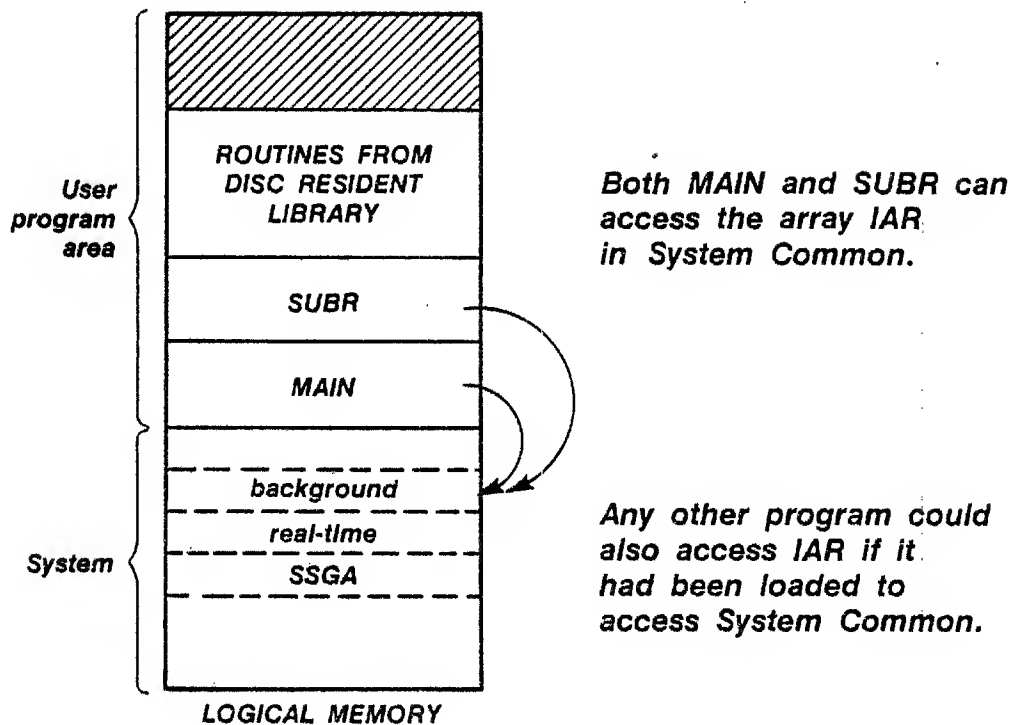
.CFER 45265 45342 750701 24998-16001

4 PAGES RELOCATED 4 PAGES REQ'D NO PAGES EMA NO PAGES MSEG

LINKS:BP PROGRAM:BG LOAD:TE COMMON:SC

/LOADR:MAIN READY AT 1:46 PM THU., 1 MAY, 1980

/LOADR:\$END



SYSTEM COMMON

- allows program to program communication
- is external to the program
- is always resident in memory (it's never swapped)



COMMANDS TO CONTROL LOADR OPERATION

/A Terminates the LOADR

LL, namr Directs the load map to "namr"

FM, DB Appends the DBUGR to the program

MP
FM, CP Selects linking mode
BP



INVOKING LOADR

There are a variety of ways to use LOADR.

INTERACTIVELY : RU, LOADR
 /LOADR: LL, 6
 /LOADR: OP, SC
 /LOADR: RE, %PROG
 /LOADR: END
 :

IN A SINGLE RUN COMMAND

: RU, LOADR, , %PROG, 6, SC

FROM A COMMAND FILE

If file PLOAD contains —

LL, 6
OP, SC
RE, %PROG
END

Invoke LOADR with —

: RU, LOADR, PLOAD

↑
command file containing
LOADR commands

LISTING ID SEGMENTS

LOADR will list information about the ID Segments in your system and the programs currently using them.

:RU,LOADR

/LOADR: OP,LI

NAME	TY	PRIOR	LMAIN	HMAIN	LO	BP	HI	BP	SZ	EMA	MSEG	PTN	TM	COM	S-ID
ZHZAT	1	41	44000	46156		22		26						NC	
D.RTR	1	1	46156	66324		26		250						NC	
PRMPT	1	5	66324	67316		250		262						NC	
\$YCOM	1	10	67316	67511		262		262						NC	
EXTND	1	10	67511	67737		262		265						NC	
UPLIN	1	3	67737	71265		265		305						SS	
QUEUE	1	2	71265	71534		305		307						SS	
GRPM	1	4	71534	72372		307		331						SS	
RTRY	1	20	72372	72620		331		332						SS	
SRQ.P	1	30	72620	72734		332		335						SS	
QUEZ	1	2	72734	73025		335		336						SS	
TTYEV	1	2	73025	73037		336		337						NC	
LGTAT	1	41	73037	75213		337		362						NC	
AUTOR	2	1	40000	41151		2		13	2				PE	NC	0
SMP	2	30	40000	50737		2		315	6				PE	NC	0
JOB	2	30	40000	51400		2		177	6				PE	NC	0
/////	3	99	40000	44150		2		207	4				PE	NC	0
FMGR	3	90	40000	46003		2		60	10				PE	NC	0
LOGON	3	50	40000	64035		2		425	12				PE	NC	0
LGOFF	3	90	40000	60523		2		377	10				PE	NC	0
R\$PN\$	3	5	40000	44420		2		46	4				PE	NC	0
GASP	3	80	40000	51107		2		135	10				PE	NC	0

This information can also be listed by —

:RU,LOADR,,,lu,LI

or

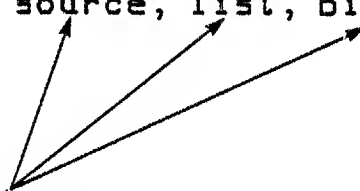
:RU,LOADR,,program,lu,LI

5D. COMPL/CLOAD


COMPL — this utility will examine the source program and run the appropriate compiler or assembler.

:RU,COMPL, source, list, binary, control statement

same as
FTN4 or ASMB



control statement replaces
that in the program



CLOAD — this utility will examine the source program and run the appropriate compiler or assembler (as does COMPL) and then run the LOADR.

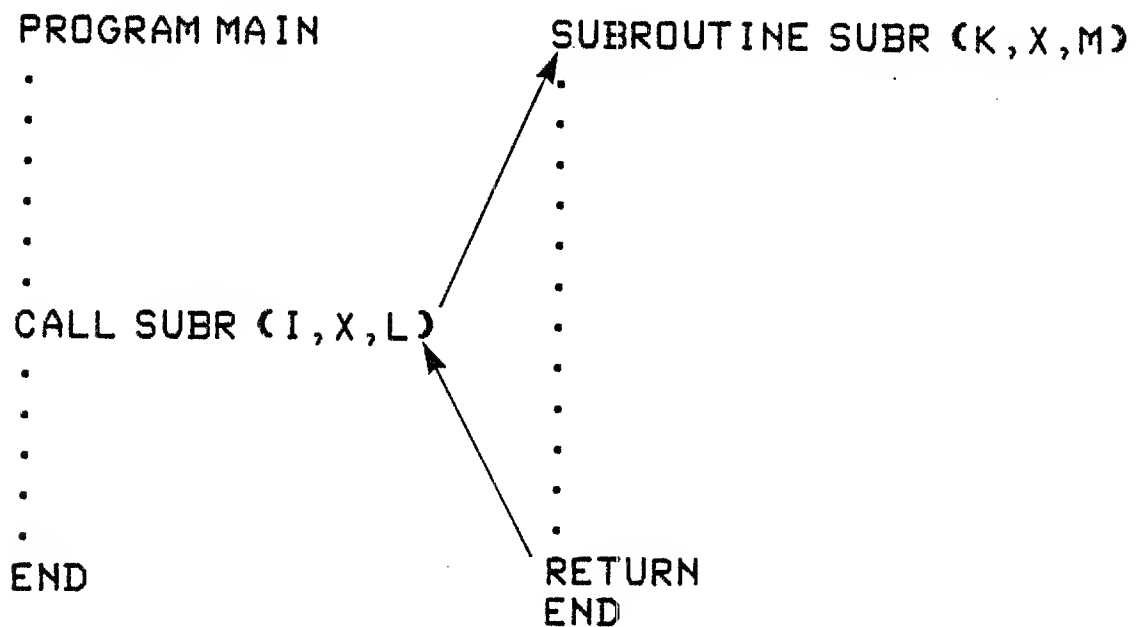
:RU,CLOAD, source, list, binary, control statement

SECTION

A	WHAT IS A PROCEDURE FILE	6-3
B	GENERALIZED PROCEDURE FILES	6-9
C	NESTED PROCEDURE FILES	6-28
D	INTERACTING WITH PROCEDURE FILES	6-30

6A. WHAT IS A PROCEDURE FILE?

Procedure Files are very similar to subprograms. A main program and a subprogram might be structured like this.



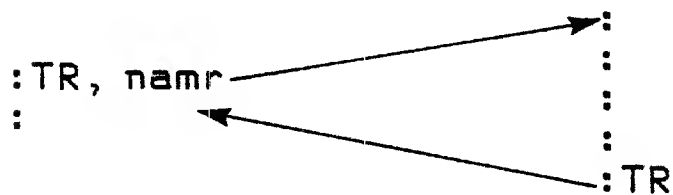
PROCEDURE FILES

A PROCEDURE FILE is:

1. a series of FMGR commands stored in a file or on a device.
2. designed to execute a frequently used series of FMGR commands.

commands entered
at a terminal

commands in a file



The FMGR TR command will invoke (transfer control to) a specified procedure file.

A TR command at the end of the procedure file will return control back to your terminal.

The commands in the procedure file will be echoed on the log device as they are executed.

A PROCEDURE FILE EXAMPLE

commands entered
at a terminal

commands in file /MAIN

:TR,/MAIN	→	:RU,FTN4,&MAIN,6,%MAIN
:	→	:RU,LOADR,,%MAIN,6
	→	:TR

The procedure file /MAIN can be used whenever you wish to compile and load the program in &MAIN.

FMGR COMMENTS

You can comment a FMGR procedure file by using the FMGR ** command.

```
or      : ** comments
or      : *, comments
or      : * b comments    (b = blank)
```

When FMGR encounters a “comment” command, it ignores the command and proceeds to the next command.

```
: ** THIS IS A PROCEDURE FILE TO
: ** COMPILE AND LOAD THE PROGRAM
: ** STORED IN FILE &MAIN
: **
: RU,FTN4,&MAIN,6,%MAIN
: RU,LOADR,,%MAIN,6
: TR
```

HELLO FILES

Session Monitor allows a System Manager to create a HELLO file for each user. The HELLO file is just a personalized procedure file.

After a user successfully logs on, FMGR will automatically transfer control to that user's HELLO file (if one was specified in the user's account).

HELLO files can be used to

- display informative messages
- set up an operating environment for the user who is logging on.

For example,

```
:RU, ANALY  
:EX
```

FMGR COMMENTS

You can comment a FMGR procedure file by using the FMGR ** command.

```
      :** comments  
or  
      :*, comments  
or  
      :* ⌀ comments    (⌀ = blank)
```

When FMGR encounters a "comment" command, it ignores the command and proceeds to the next command.

```
:** THIS IS A PROCEDURE FILE TO  
:** COMPILE AND LOAD THE PROGRAM  
:** STORED IN FILE &MAIN  
:**  
:RU,FTN7X,&MAIN,6,%MAIN  
:RU,LOADR,,%MAIN,6  
:TR
```

6B. GENERALIZED PROCEDURE FILES

```
procedure file { :RU,FTN4,&MAIN,6,%MAIN  
  /MAIN         :RU,LOADR,,%MAIN,6  
                :TR
```



A GENERALIZED PROCEDURE FILE TO COMPILE AND LOAD A PROGRAM

A procedure file to compile and load any program and list to any device might be —

```
:RU,FTN4,    <unspecified> <unspecified> <unspecified>  
              source      list      relocatable  
:RU,LOADR,,  <unspecified> <unspecified>  
              relocatable list  
:TR
```

You “specify the unspecified” when invoking the procedure file.

```
:TR,/MAIN,&MAIN,6,%MAIN -  
      ^       ^       ^  
specify the specify the specify the  
  source    list    relocatable
```



JUST LIKE A SUBPROGRAM

A generalized procedure file is analogous to a subprogram with parameters.

For example,

```
          SUBROUTINE PSUM (I,J,LU)
          ISUM=I+J
          WRITE (LU, 101) ISUM
101      FORMAT (/"SUM IS", I5)
          RETURN
          END
```

Variables I, J, LU are "unspecified" values; they are assigned values when the subprogram is invoked.

The calling program might contain these statements,

```
          .
          .
          .
          K=15
          CALL PSUM (K,35,6)
          .
          .
          .
```

The unspecified parameters (I,J,LU) are then assigned values:

K → I
35 → J
6 → LU



FMGR has a set of variables, called GLOBALS, that may be set, modified or examined by FMGR commands entered interactively or via a procedure file.

GLOBALS in a procedure file serve to generalize the procedure file.

- GLOBALS represent the unspecified parameters of the commands in the procedure file.
- Values are passed to these unspecified parameters (GLOBALS) when the procedure file is invoked.

G GLOBALS

FMGR has 12 G GLOBALS called —

0G 1G 2G 3G 4G 5G 6G 7G 8G 9G 10G 11G

Each G GLOBAL can contain

1. nothing (null value)
2. an integer value
3. up to 6 ASCII characters

Only G GLOBALS 1G→9G may be set or modified by a FMGR command (interactive or in a procedure file).



PASSING VALUES TO G GLOBALS



An extended form of the TR command will invoke a procedure file and pass up to 9 values to the Globals 1G → 9G.

`:TR, namr[,v1 ,v2 ,v3 ,v4 ,v5 ,v6 ,v7 ,v8 ,v9]`

up to 9 optional values passed to 1G → 9G

- *values are passed according to position*
- *omitted values cause no change to the corresponding G GLOBAL*
- *the value may be specified as a constant or another GLOBAL*

For example,

```
:TR, /MAIN,,7,THIS,25  
:TR, /PRDG,7G
```

A GENERALIZED COMPILE AND LOAD PROCEDURE FILE

Procedure file /MAIN might be generalized as follows:

```
*** PROCEDURE FILE TO COMPILE AND
*** LOAD A PROGRAM
***
*** GLOBAL 1G REPRESENTS SOURCE FILE
***          2G REPRESENTS LIST
***          3G REPRESENTS RELOCATABLE FILE
***
:RU,FTN4,1G,2G,3G
:RU,LOADR,,3G,2G
:TR
```

/MAIN might be invoked with:

```
:TR,/MAIN,&PROGA,1,%PROGA
      ↓      ↓      ↓
      1G    2G    3G
```



SPECIAL G GLOBALS

G GLOBALS 0G and 10G have special uses:

- 0G contains FMGR's input device LU
- 10G LOADR places the name of the program
 just loaded into 10G

For example, /MAIN can be modified to compile, load and run a program:

```
:RU,FTN4,1G,2G,3G
:RU,LOADR,,3G,2G
:RU,10G
:TR
```

/MAIN might be invoked with:

```
:TR,/MAIN,&FILE,0G,%FILE
```

DISPLAYING GLOBAL VALUES

The FMGR DP (DISPLAY) command will display constants, character strings and values of GLOBALS on the log device.

:DP, x1, x2, x3 . . .

parameters to be displayed

for example

```
:DP,17,0G,THIS IS A STRING,10G
17,1, THIS IS A STRING, PROGX
```

You can also use the FMGR SE command to assign values to G GLOBALS.

: SE, $x_1, x_2, x_3, x_4, \dots, x_9$
 values to be assigned to $1G \rightarrow 9G$

- position determines which G GLOBAL is assigned a value
- omitted values do not affect the value of the corresponding G GLOBAL
- values may be constants, character strings or other GLOBALS

```

For example,      :SE,17,-13,FTN4,2000B
                  :DP,1G,2G,3G,4G

                  :SE,, -15
                  :DP,1G,2G

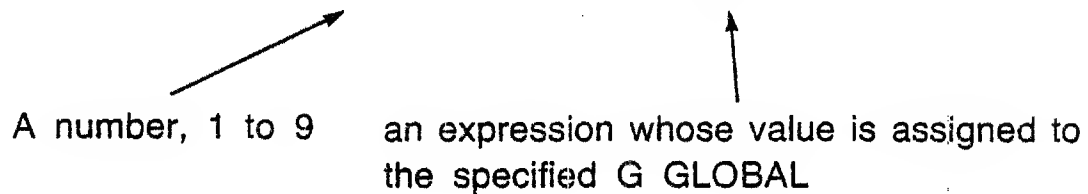
                  :SE,2G
                  :DP,1G,2G

```


CALCULATING G GLOBALS

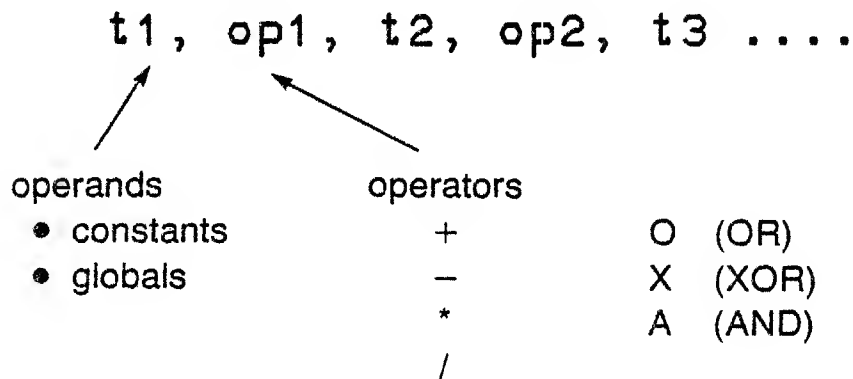
A third way to assign a value to a G GLOBAL is with the FMGR CA command.

:CA, global number, expression


A number, 1 to 9 an expression whose value is assigned to
the specified G GLOBAL

The expression is of the form:

t1, op1, t2, op2, t3


operands operators

- constants
- globals

+	O (OR)
-	X (XOR)
*	A (AND)
/	

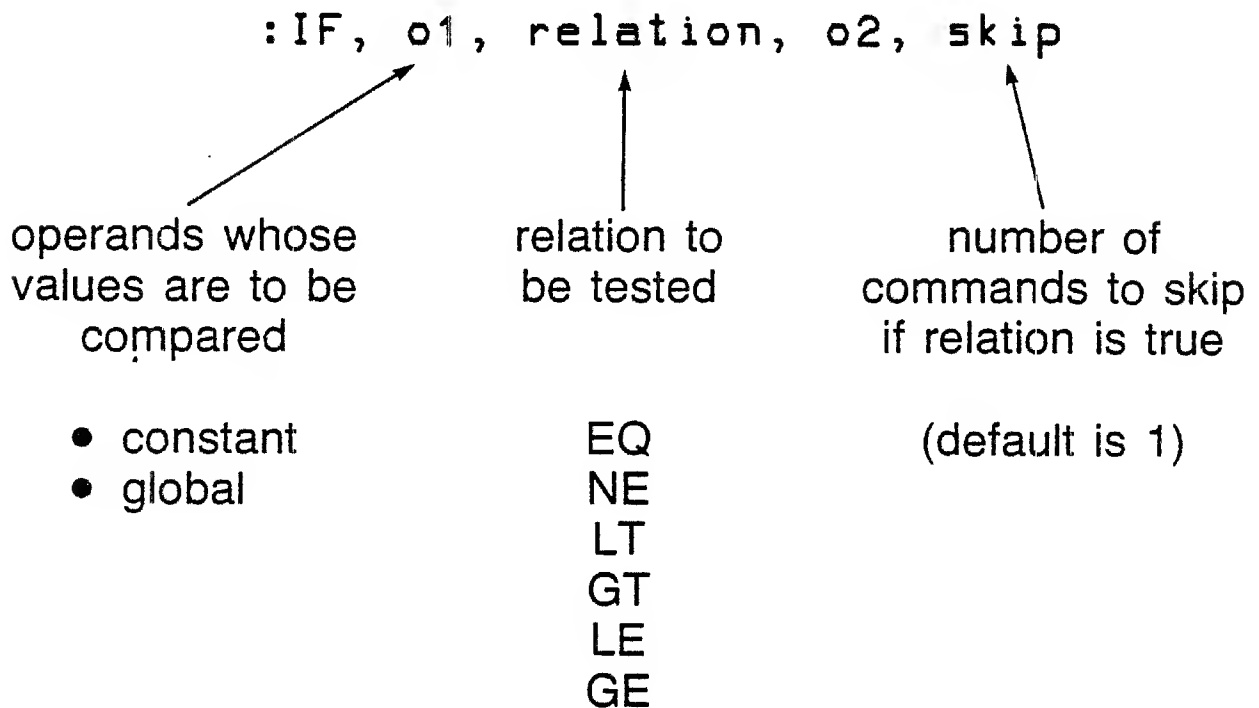
for example,

:SE, 3
:SE, , , 15
:CA, 9, 5, +, 3G, /, 2
:DP, 1G, 3G, 9G

:CA, 8, 150
:DP, 8G

DECISIONS AND LOOPS IN PROCEDURE FILES

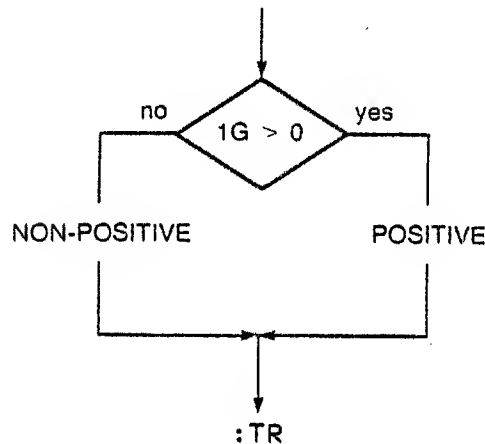
The FMGR IF command allows you to alter the normal flow of control in a procedure file.



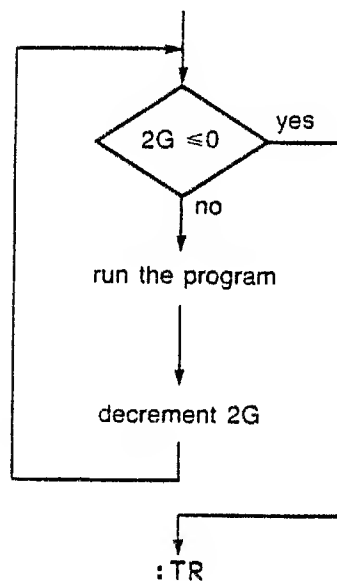
THE IF COMMAND MAY NOT BE USED INTERACTIVELY!

IF COMMAND EXAMPLES

A DECISION Procedure file /DEC is to be invoked with one value passed to it (in 1G). /DEC should examine the value and print either POSITIVE or NON-POSITIVE.



A LOOP Procedure file /LOOP is to be invoked with two values passed to it. 1G is a program name; 2G is the number of times 1G should be run.



INTERNAL G GLOBAL STRUCTURE

G Type Globals are stored as four words each as follows:

0	TYPE
1	DATA 1
2	DATA 2
3	DATA 3

where:

TYPE = 0 if the global is null

= 1 if DATA 1 is an integer

= 3 if DATA 1 → DATA 3 are ASCII characters

GLOBAL FORMAT

WORD 0	GLOBAL TYPE=0 (NULL)	GLOBAL TYPE=1 (NUMERIC)	GLOBAL TYPE=3 (ASCII)
WORD 1	0	INTEGER	CHARACTERS 1,2
WORD 2	0	0	CHARACTERS 3,4
WORD 3	0	0	CHARACTERS 5,6

GLOBAL EQUIVALENCE

G	P
0	-40 Type
	-39
	-38
	-37
1	-36 Type
	-35
	-34
	-33
2	-32 Type
	-31
	-30
	-29
3	-28 Type
	-27
	-26
	-25
4	-24 Type
	-23
	-22
	-21

G	P
5	-20 Type
	-19
	-18
	-17
6	-16 Type
	-15
	-14
	-13
7	-12 Type
	-11
	-10
	- 9
8	- 8 Type
	- 7
	- 6
	- 5
9	- 4 Type
	- 3
	- 2
	- 1
10	0 Type
	1
	2
	3
11	4
	5
	6
	7
	8
	9



P GLOBALS may be examined, set or modified exactly like G GLOBALS, except that they refer to one word only and are interpreted as integer values.

You can use the FMGR CA (CALCULATE) command to assign values to the P GLOBALS —

-36 P to -1P (correspond to 1G → 9G)
or
1P to 5P

The command format is —

:CA, n:P, expression

n can be —

-36 to -1	an expression whose value
or	is assigned to the specified
1 to 5	P global

P GLOBAL EXAMPLES

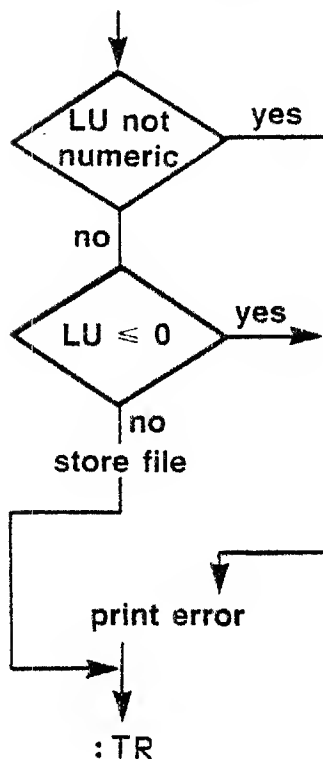
Example 1

```
:SE,15,,14
:CA,3:P,1G,-,3G
:DP,3P
```

Example 2

```
:SE,,,,15
:CA,-23:P,5
:DP,-23P,4G
```

Example 3 Procedure file /DUMP will read a file from a mag tape and store it into a disc file. 1G is used for the mag tape LU; 2G is used for the disc file. /DUMP should check to see if the mag tape LU specified is a positive numerical value.



SPECIAL USES OF P GLOBALS

FTN4 P GLOBALS 1P to 5P are set to values reflecting the number of errors found in the source program just compiled.

1P	total number of disasters, errors and warnings
2P	number of disasters
3P	number of errors
4P	number of warnings
5P	revision number of the compiler

LOADR For a successful load —

1P	}	program name
2P		
3P		
4P	}	spaces
5P		

For an unsuccessful load —

1P	}	6 character mnemonic error code
2P		
3P		
4P	L-	
5P	0	

SPECIAL P GLOBALS

6P—CONTAINS LAST ERROR NUMBER FROM FMGR

7P—CONTAINS CURRENT SEVERITY CODE

8P—CONTAINS YOUR SESSION IDENTIFIER

9P—CONTAINS YOUR CAPABILITY LEVEL

EXAMPLE

:CR, XYZ:::4:1

:CR,XYZ:::4:1

FMGR-002

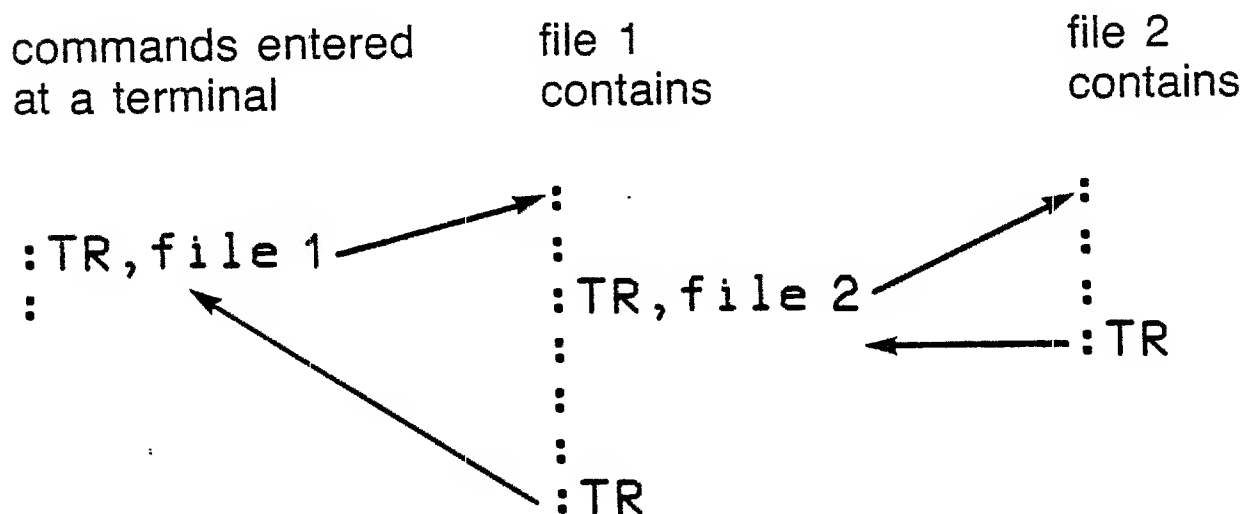
:DP,6P

-2

:

6C. NESTED PROCEDURE FILES

Procedure files may invoke other procedure files just as subprograms may invoke other subprograms.



FMGR keeps track of nested procedure files with a TRANSFER STACK. The Transfer Stack may contain up to 10 entries.

MORE ON THE TR COMMAND

Additional forms of the FMGR TR command allow various means of controlling the nesting of procedure files.

- invoke a procedure file

:TR, namr, parameters

- transfer back one procedure file in the stack

:TR

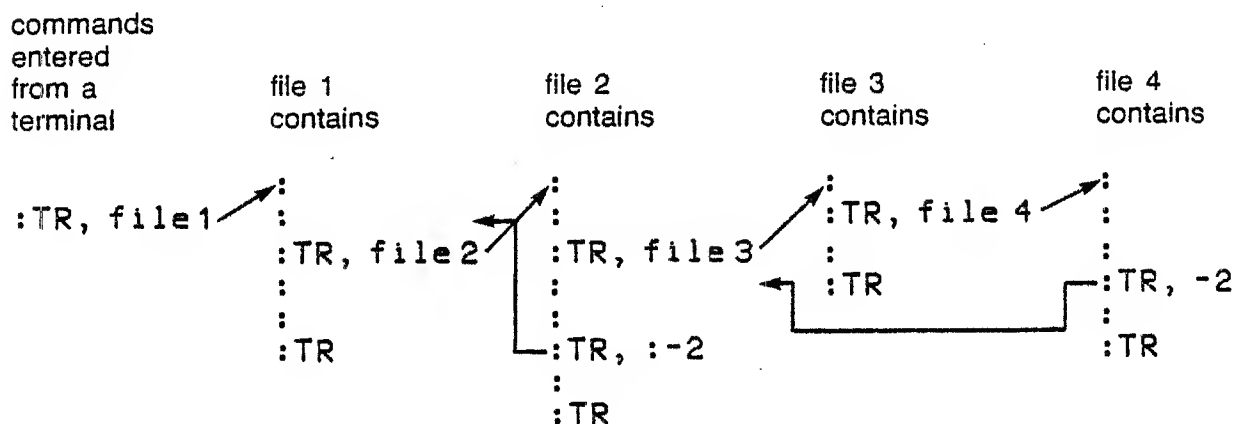
- transfer back a specified number of files in the stack

:TR, - integer

- transfer back to the previous procedure file, but go back the specified number of commands in that procedure.

:TR, : - integer

For example



6D. INTERACTING WITH PROCEDURE FILES

MESSAGES

You can use FMGR commands to send messages to be printed at the:

LOG DEVICE, use the FMGR DP (DISPLAY) command

:DP, message

LIST DEVICE, use the FMGR AN (ANNOTATE) command

:AN, message

SYSTEM CONSOLE, use the FMGR TE (TELL) command

:TE, message

SUSPENDING AND RESTARTING A PROCEDURE FILE

The execution of a procedure file may be suspended and control transferred to an interactive device with the FMGR PA (PAUSE) command.

```

from a procedure file          at the specified LU
                                (default is log device)

:                               : PA, lu, message (the PA
:                               :                               command
:                               :                               is echoed)
: PA, lu, message              :
:                               :
:                               : (you then interactively
:                               : enter commands
:                               : as desired)
:                               :
:                               : TR (transfer back to
:                               : procedure file)

```

ERRORS IN PROCEDURE FILES

When FMGR encounters an error while executing a procedure file, control is transferred to the log device so corrective action may be taken.

procedure file

log device (interactive)

:			
:			
:		FMGR-006	(file not found)
:		:ST,8,FILEA	
:LI, FILEA	←	:TR, :-1	you then take
:LI, FILEB			corrective action
:			and transfer
:			back to the
:TR			procedure file

By setting FMGR's SEVERITY CODE you can control

- echoing of commands
- listing of errors
- transferring to the log device on errors

7

USING RTE's SERVICES PROGRAMMATICALLY



SECTION

A INTRODUCTION TO EXEC CALLS 7-3

B I/O PROCESSING

7A. INTRODUCTION TO EXEC CALLS

You call EXEC to request RTE services as follows:

CALL EXEC (ICODE, p1, p2, p3, ... pn)

a request code, identifies
the service requested

parameters further specifying
the request

For example,

An EXEC 11 (ICODE = 11) is a request for RTE's current time of day.

```
DIMENSION ITIME(5)
.
.
.
.
ICODE = 11
CALL EXEC (ICODE, ITIME[, IYEAR])
```

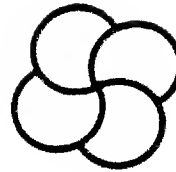
RTE places the

- day
- hour
- minute
- second
- 10's of milliseconds
(centiseconds)

in this array.

RTE returns the
4 digit year in
this variable

SUCCESS?



A request for an RTE service might be:

SUCCESSFUL

- control is returned to the statement following the EXEC call
- status information may be available in the A and B registers

UNSUCCESSFUL

- your program will usually be aborted and an error message will be displayed at your terminal (and at the system console)



RETRIEVING INFORMATION FROM THE A and B REGISTERS

An EXEC 1 request is for input —

```
.  
.  ICODE = 1  
  CALL EXEC (ICODE,1,I,1)  
  CALL ABREG (IA,IB)
```

The library routine ABREG stores the
A register contents in the first
parameter, the B register contents
in the second parameter

A successful READ request will
return with:
A register: device status
B register: number of words
(characters)
transferred

You should call ABREG immediately after the call
to EXEC.



:RU,TMP

ENTER AN INDEX VALUE:

IO12 TMP 40237

TMP ABORTED

ABEND TMP ABORTED

:HE

IO12

AN I/O REQUEST SPECIFIED A LOGICAL UNIT NOT DEFINED FOR USE BY
THIS SESSION. THE "SL" COMMAND WILL REPORT ALL LOGICAL UNITS
AVAILABLE TO YOUR SESSION.

:

DEBUGGING EXEC ERRORS

The compiler listing and load map for program TMP are —

```

0001  FTN4,L,Q
0002  00000      PROGRAM TMP
0003  00000      INTEGER ARRAY(100)
0004  00000 C
0005  00000 C      INITIALIZE ARRAY 1 TO 100, BACKWARDS
0006  00000 C
0007  00000      DO 10 I = 1,100
0008  00151 10          ARRAY(101-I) = I
0009  00151 C
0010  00167 20      CONTINUE
0011  00167      WRITE(1,101)
0012  00175 101      FORMAT(/"ENTER AN INDEX VALUE:")
0013  00175      CALL EXEC(1,15,INDEX,1)
0014  00203      CALL ASCBN(INDEX)
0015  00206      IF(INDEX .EQ. 0) GOTO 99
0016  00215          IVAL = ARRAY(INDEX)
0017  00221          WRITE(1,102) IVAL
0018  00231 102      FORMAT(/"THE CORRESPONDING ARRAY ELEMENT IS ",I5)
0019  00231          GOTO 20
0020  00231 C
0021  00231 99      CONTINUE
0022  00232      END

```

:RU,LOADR

```

/LOADR:  RE,%TMP
        TMP      40042 40354
/LOADR:  RE,%ASCBN
        ASCBN    40355 40502
/LOADR:  END

```

```

FMTIO  40503 42001      24998-16002 REV.1926 790417
FMT.E   42002 42002      24998-16002 REV.1901 781107
PNAME   42003 42050      771121 24998-16001
REIO    42051 42175      92067-16268 REV.1903 790316
FRMTR   42176 45633      24998-16002 REV.1926 790503
.CFER   45634 45711      750701 24998-16001

```

```

  4 PAGES RELOCATED      4 PAGES REQ'D      NO PAGES EMA      NO PAGES MSEG
LINKS:BP      PROGRAM:BG      LOAD:TE      COMMON:NC
/LOADR:TMP      READY AT  2:24 PM  THU.,  1  MAY , 1980

```

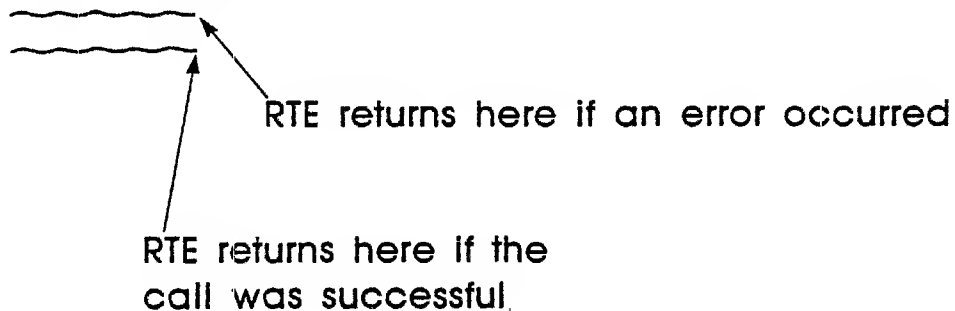
/LOADR:\$END

HANDLING EXEC ERRORS PROGRAMMATICALLY

For less severe errors, a user can specify that a program should not be aborted if an EXEC error occurs. If this option is selected however, the program should process an error itself.

The "no abort" option is selected by setting bit 15 of the ICODE parameter of an EXEC call; the "no abort" option is then in effect for that particular call.

```
ICODE = 1 + 100000B  
CALL EXEC (ICODE,1,I,1)
```



The error return point should always contain a GO TO statement.

RETRIEVING ERROR INFORMATION

When an EXEC error occurs, RTE stores error information in the A and B registers.

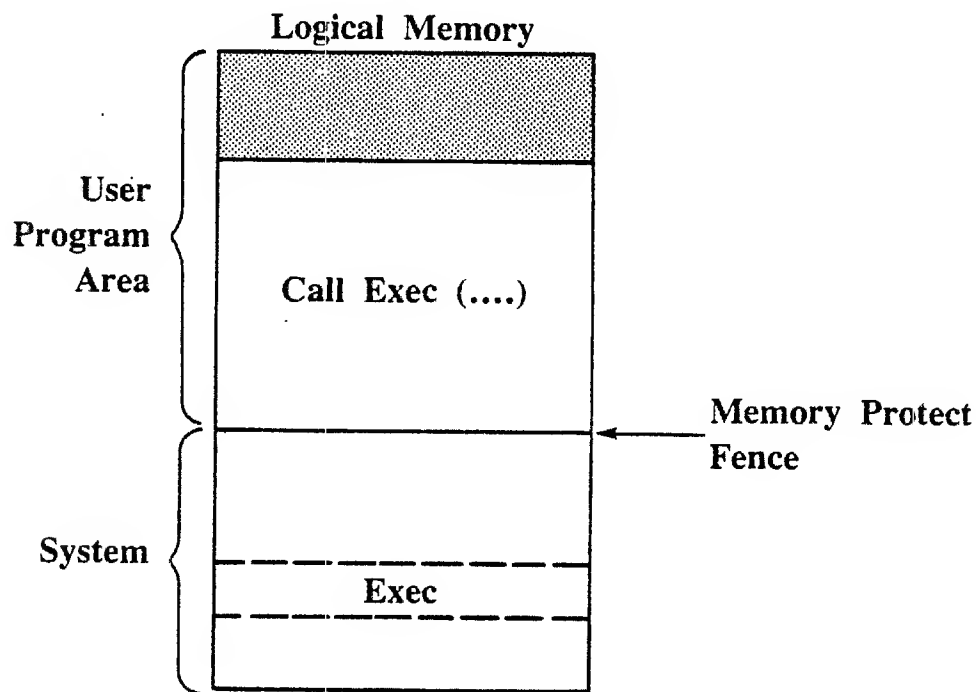
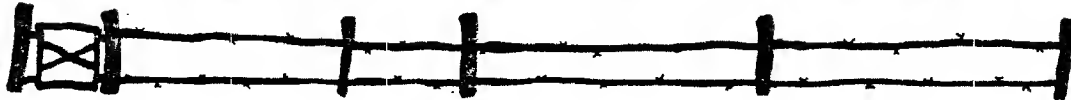
*A register – 2 character (ASCII) error code: SC,
LU, IO, DR or RN*

*B register – 2 digit (ASCII) error number: 01,
02, 03*

For example,

```
.  
. .  
. .  
    ICODE = 1 + 100000B  
    CALL EXEC (ICODE,1,I,1)  
    GO TO 99  
10   WRITE(1,101)I  
    .  
    .  
    .  
    .  
99   CALL ABREG(IA,IB)  
    WRITE(1,201)IA,IB  
201  FORMAT(/"EXEC ERROR ON INPUT:"/  
    *      "ERROR CODE",2A2)  
    STOP  
    .  
    .  
    .
```

MEMORY PROTECT FENCE

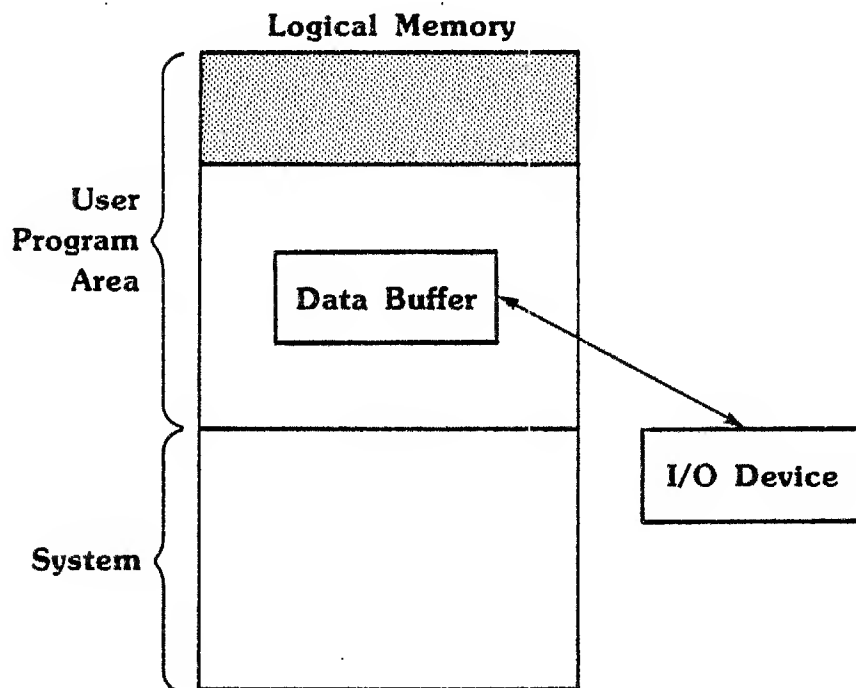


RTE has a **MEMORY PROTECT FENCE** to protect itself from user programs. User programs crossing the fence will be aborted with an MP error.

7B. I/O PROCESSING

I/O requests made with FORTRAN READ/WRITE statements are first processed by the FORTRAN FORMATTER and then sent to RTE for the actual I/O operation.

By using EXEC I/O requests, you can request I/O operations directly.



EXEC READS & WRITES

EXEC READ (EXEC 1) — inputs data from a device into a buffer in your program

```
ICODE = 1  
CALL EXEC (ICODE, ICNWD, IBUFF, ILEN)
```

EXEC WRITE (EXEC 2) — outputs data from a buffer in your program to a device

```
ICODE = 2  
CALL EXEC (ICODE, ICNWD, IBUFF, ILEN)
```

where ICODE — the EXEC request

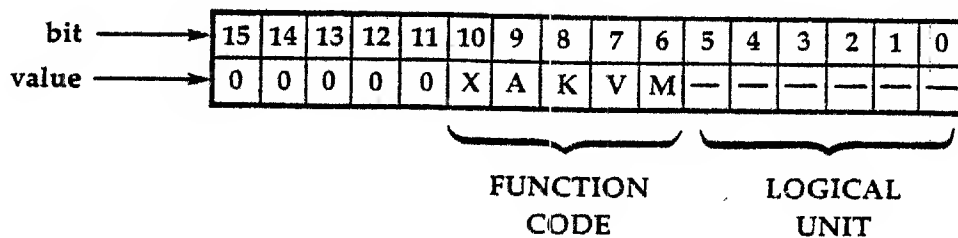
ICNWD — control word : specifies "how and where" to perform the I/O operation

IBUFF — array in the program acting as the data buffer

ILEN — positive number of words or negative number of characters to be transferred

ICNWD

ICNWD is a control word containing information that tells RTE "how and where" to perform the data transfer.



WHERE:

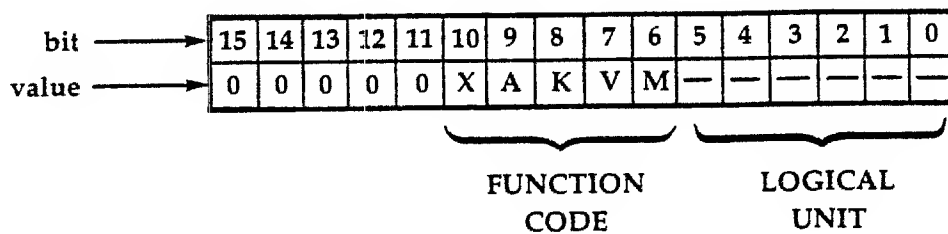
Logical Unit = LU number of device to use in transfer. "Where to perform the I/O operation"

Function Code = control bits used to control device driver (device driver dependent).
 "How to perform the I/O operation"

0 = bits used by the system that should be set to 0.

FUNCTION CODE

The function code field of ICNWD is used to control the device driver. The meaning of the bits depends on the driver type.



For example:

M = 0 data is ASCII
 1 data is binary

} for a CTU READ or WRITE

V = 0 use first column as carriage control
 1 print the first column

} for a line printer

K = 0 no echo on an EXEC read
 1 echo on an EXEC read

} for a terminal

A = 0 terminal enabled character/block read
 1 program enabled block read

} for a terminal

X = 0 use V bit for control information
 1 user supplies CR, LF, FF controls

} for a line printer

AN EXAMPLE OF EXEC READS & WRITES

```
0001  FTN4,L
0002      PROGRAM TEXEC
0003      INTEGER IBUF(5)
0004  C
0005  C      THIS PROGRAM REQUESTS UP TO 10 CHARACTERS FROM THE
0006  C      TERMINAL (LU 1) AND PRINTS THEM ON THE LINE
0007  C      PRINTER (LU 6).
0008  C
0009      WRITE(1,101)
0010  101  FORMAT(/"PLEASE TYPE UP TO 10 CHARACTERS:")
0011  C
0012  C      EXEC READ TO RETRIEVE INPUT,
0013  C      SET THE "K" BIT FOR ECHO.
0014  C
0015      ICNWD = 1 + 400B
0016      CALL EXEC(1, ICNWD,IBUF,-10)
0017  C
0018  C      GET THE NUMBER OF CHARACTERS ACTUALLY ENTERED.
0019  C
0020      CALL ABREG(IA,IB)
0021      ILEN = IB
0022  C
0023  C      EXEC WRITE TO PRINT THE STRING,
0024  C      SET THE "V" BIT TO PRINT THE FIRST CHARACTER.
0025  C
0026      ILEN = -ILEN
0027      ICNWD = 6 + 200B
0028      CALL EXEC(2, ICNWD,IBUF,ILEN)
0029  C
0030      END
```

EXEC DEVICE CONTROL

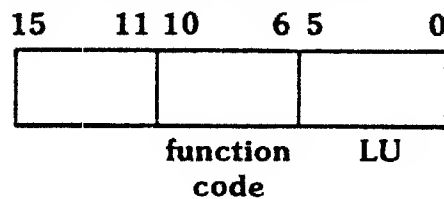
An EXEC 3 call will allow you to programmatically control I/O devices, for example

- enable/disable a terminal
- rewind a mag tape
- issue a form feed to the line printer

```
ICODE = 3
CALL EXEC (ICODE, ICNWD, IOP1)
```

where

ICNWD — control word specifying the control function and LU



↑
specifies control function to be performed

IOP1 — optional parameter, required by some control functions for extra information

➤OBTAINING DEVICE STATUS➤

- **DEVICE STATUS AFTER AN EXEC READ/WRITE**

After a successful EXEC 1 or 2 request, the A register contains the device status (Eqt word 5).

- **DEVICE STATUS WITH AN EXEC 13 REQUEST**

The EXEC 13 call will return information about a device, including the device type and its last reported status.

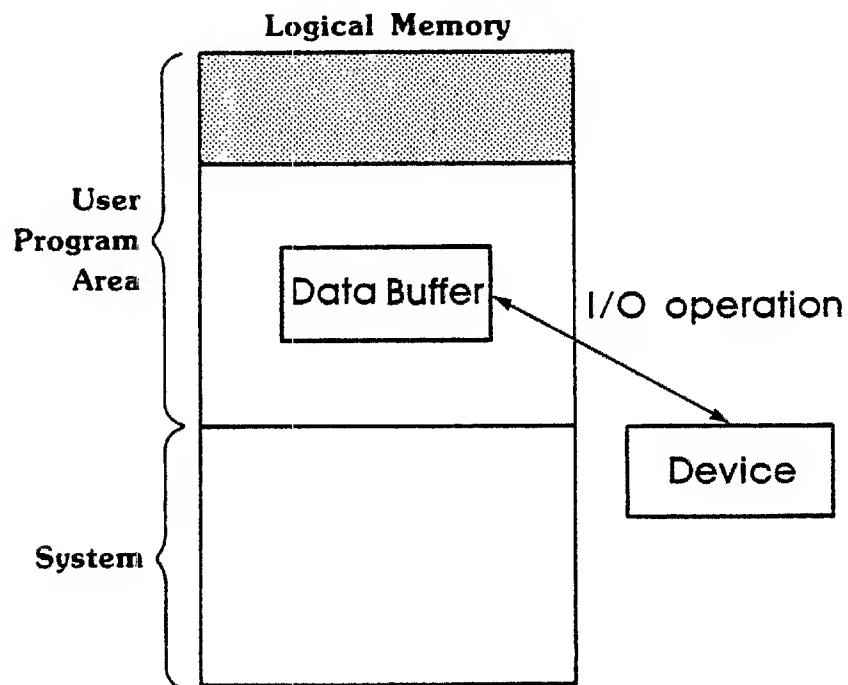
```
ICCODE = 13  
CALL EXEC (ICCODE, ICNWD, IST1, IST2, IST3)
```

where

```
ICNWD  — LU of device being queried  
IST1   — returned with word 5 of EQT  
IST2   — returned with word 4 of EQT  
IST3   — returned with subchannel of device and "up"  
         or "down" information
```

NORMAL I/O

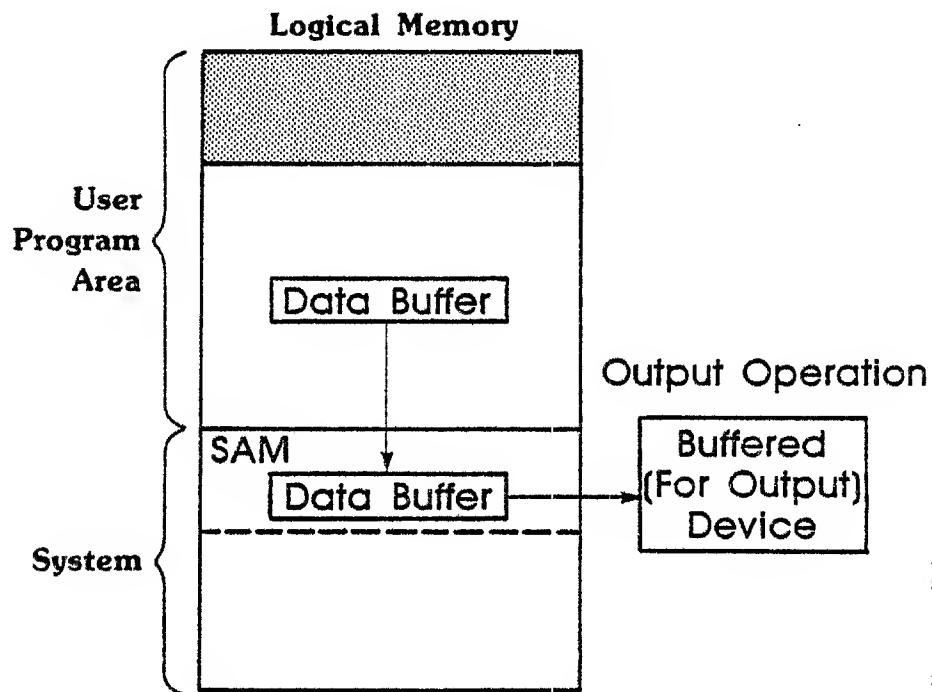
A program doing EXEC READS or WRITES to a (unbuffered) device is requesting a data transfer directly between the device and a buffer in the user program area of memory.



AUTOMATIC OUTPUT BUFFERING

Some devices (such as terminals or line printers) may be specified to be **BUFFERED** devices for output operations.

This specification is done when the RTE system is generated or by using the system EQ command (high capability).



SYSTEM AVAILABLE MEMORY (SAM) is a dynamic data area in the system area of memory.

REENTRANT I/O

The library routine REIO will perform I/O operations such that:

- the program becomes swappable

```
CALL REIO( ICODE, ICNWD, IBUFF, ILEN )
```

same as for EXEC 1 or 2
(only non-disc devices)

REIO will always perform the requested I/O operation; however, the program will be swappable if

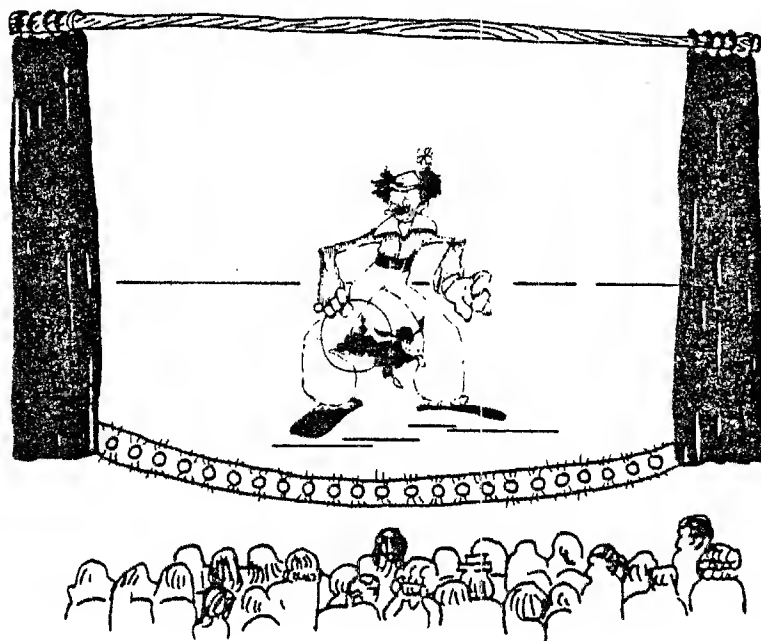
- ILEN is less than 130 words
- IBUFF is at least 5 words from the beginning of your program

REIO uses a data buffer in SAM to hold the data from your program so that your program may be swapped if needed.

I/O SUMMARY

	PROGRAM STATE	SWAPPABLE?
FORTRAN		
READ WRITE		

8 INTERACTING WITH YOUR PROGRAM



SECTION

A	PASSING INFORMATION	8-3
B	SUSPENDING PROGRAMS	8-9
C	TERMINATING PROGRAMS	8-12

8A. PASSING INFORMATION

Values may be passed to

- * subprograms when you call them

```
CALL SUBR (I,J,K)
```

- * procedure files when you transfer to them

```
:TR,/MAIN,&MAIN,6,%MAIN
```

- * programs when you run them

```
:RU,LOADR,,%MAIN,6
```

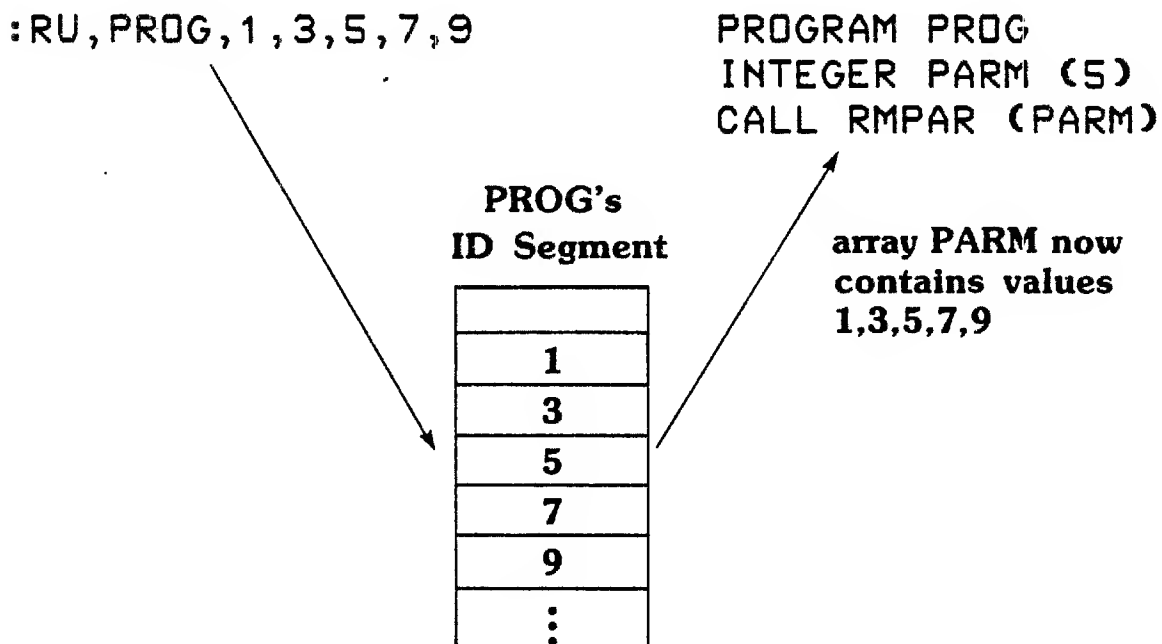


RMPAR

When a program is run, up to 5 integer values (or pairs of ASCII characters) may be included as parameters in the RU command.

```
S= xx COMMAND ?RU,KYDMP,KF,IL,E1
      :RU,LISTR,6,TE,XT,F4
```

The program can then use the library routine RMPAR to retrieve these values and store them in an array in the program.



The call to RMPAR should be the first executable statement in your program.

RETURNING VALUES TO FMGR FROM A PROGRAM

If you run a program with the FMGR RU command, your program can return up to 5 integer values (or ASCII character pairs) to FMGR. FMGR retrieves these values and stores them in globals 1P to 5P.

The library routine PRTN is used to return values, for example:

```
0001  FTN4,L
0002      PROGRAM INOUT
0003  C
0004  C      THIS PROGRAM IS TO BE RUN WITH 5 VALUES PASSED
0005  C      TO IT.  THE PROGRAM WILL RETRIEVE THE 5 VALUES
0006  C      AND THEN PASS THEM BACK TO FMGR IN REVERSE ORDER.
0007  C
0008      INTEGER IPARM(5) , RPARM(5)
0009  C
0010      RETRIEVE THE PASSED VALUES
0011  C
0012      CALL RMPAR(IPARM)
0013  C
0014      DO 10 I = 1,5
0015 10      RPARM(I) = IPARM(6-I)
0016  C
0017  C      RETURN THE VALUES TO FMGR
0018  C
0019      CALL PRTN(RPARM)
0020  C
0021      END
```

```
:RU,INOUT,1,2,3,4,5
:DP,1P,2P,3P,4P,5P
 5,4,3,2,1
:RU,INOUT,,,UP,DN
:DP,10G
DNUP
:
```


PASSING STRINGS TO PROGRAMS

Strings of characters may also be passed to a program, for example

:RU,GRSTR,THIS IS A STRING

string of characters to be
passed to GRSTR

A program can retrieve a string via

- EXEC 14 — the EXEC 14 will retrieve
REQUEST the “run string”

:RU,GRSTR,THIS IS A STRING

EXEC 14 will retrieve this

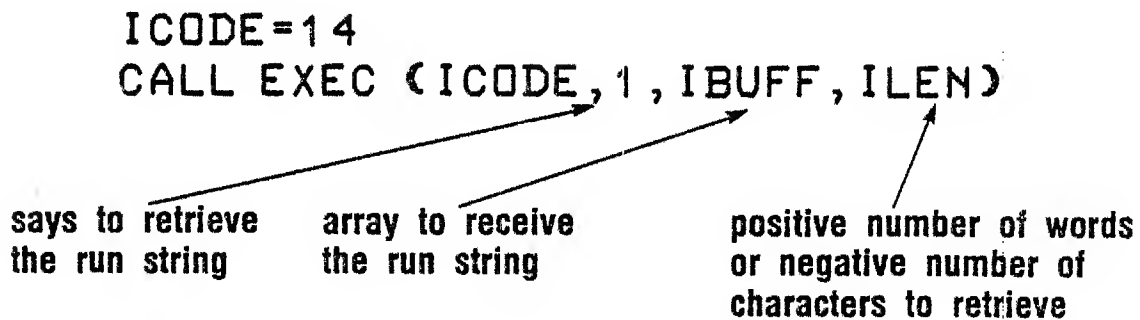
- GETST — the library routine GETST
will retrieve the “parameter
string” (any characters after
the second comma)

:RU,GPSTR,THIS IS A STRING

GETST will retrieve this

RETRIEVING THE RUN STRING

The EXEC 14 will retrieve the command that scheduled the program.



For example,

```

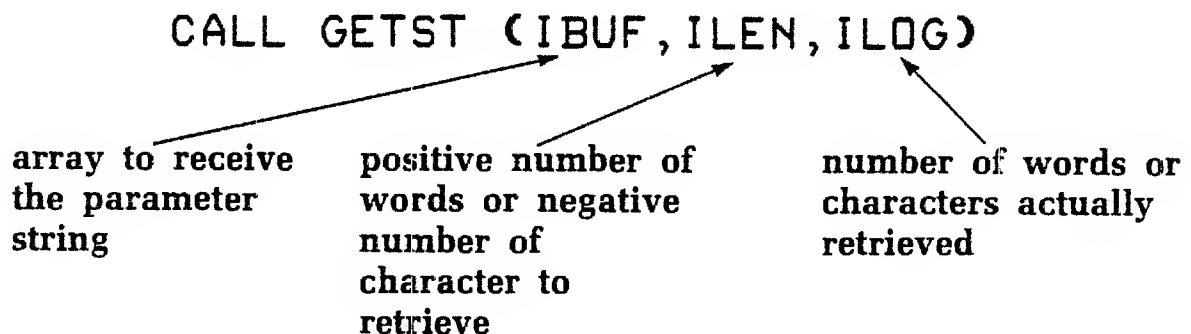
0001  FTN4,L
0002      PROGRAM GRSTR
0003      INTEGER IBUF(35)
0004  C
0005  C      RETRIEVE THE RUN STRING VIA EXEC 14
0006  C      SPECIFY THE MAX NUMBER OF WORDS TO BE RETRIEVED.
0007  C
0008      ILEN = 35
0009      ICODE = 14
0010      CALL EXEC(ICODE,1,IBUF,ILEN)
0011      CALL ABREG(IA,IB)
0012      ILOG = IB
0013  C
0014  C      PRINT THE RUN STRING,
0015  C      USING THE ACTUAL NUMBER OF WORDS RETRIEVED.
0016  C
0017      WRITE(1,101) (IBUF(I),I=1,ILOG)
0018 101  FORMAT(/"THE RUN STRING IS:"/
0019      1 35A2)
0020      END
  
```

:RU,GRSTR,THIS IS A STRING

THE RUN STRING IS:
 RU,GRSTR,THIS IS A STRING
 :

RETRIEVING THE PARAMETER STRING

A call to GETST will retrieve the parameter string part of the command that scheduled the program.



For example,

```
0001  FTN4,L
0002      PROGRAM GPSTR
0003      INTEGER IBUF(35)
0004  C
0005  C      RETRIEVE THE PARAMETER STRING VIA GETST,
0006  C      SPECIFY THE MAX NUMBER OF WORDS TO BE RETRIEVED.
0007  C
0008      ILEN = 35
0009      CALL GETST(IBUF,ILEN,ILOG)
0010  C
0011  C      PRINT THE PARAMETER STRING,
0012  C      USING THE ACTUAL NUMBER OF WORDS RETRIEVED.
0013  C
0014      WRITE(1,101) (IBUF(I),I=1,ILOG)
0015 101  FORMAT(/"THE PARAMETER STRING IS:"/
0016      1 35A2)
0017      END
```

```
:RU,GPSTR,THIS IS A STRING
```

```
THE PARAMETER STRING IS:
THIS IS A STRING
```

```
:
```

8B. SUSPENDING PROGRAMS — INTERACTIVELY —

You can **INTENTIONALLY** suspend a program and then restart its execution by using the system **SS** and **GO** commands.

S=xx COMMAND? SS,program

↑
“program” will be suspended

S=xx COMMAND? GO,program

↑
“program” will be rescheduled

(If “program” is not specified,
the current Session program is
suspended or rescheduled)

SUSPENDING PROGRAMS — PROGRAMMATICALLY —

A program can suspend itself and then be rescheduled by an operator via

- issuing a READ request and waiting for the operator to respond.
- executing a FORTRAN PAUSE statement and waiting for the operator to enter a "GO,program" command.

PAUSE xxxx

↑
octal value displayed in a PAUSE
message at your terminal

- making an EXEC 7 request and waiting for the operator to enter a "GO,program" command.

```
ICODE=7  
CALL EXEC(ICODE)
```

PASSING VALUES WHEN RESTARTING A PROGRAM



The GO command may pass up to 5 integer values (or ASCII character pairs) to the program that is being restarted. RMPAR is used by the program to retrieve the values.

```
PROGRAM PROGA  
INTEGER PARM(5)
```

```
...
```

```
CALL EXEC(7)
```

```
CALL RMPAR (PARM) ←
```

↑
PARM then contains
5,6,1,3,8

8C. TERMINATING PROGRAMS

A program may be terminated in several ways:

- * FORTRAN STOP statement

STOP xxxx

↑
octal value displayed in a STOP message
at your terminal

- * EXEC 6 REQUEST

CALL EXEC(6)

- * FORTRAN END statement

END

The END statement causes the FTN4 compiler to generate an EXEC 6 request automatically.



TERMINATING A PROGRAM EARLY

If you want to terminate a program before its normal termination —

- use the system OF command

S=xx COMMAND? OF,program $\begin{bmatrix} ,0 \\ ,1 \\ ,8 \end{bmatrix}$
or

S=xx COMMAND? OF

- your program may be created to allow an early termination by examining its "break bit".

PROGRAM BREAKS

One of the bits in a program's ID segment is the "break bit". The library routine IFBRK allows a program to check its break bit (and clear it if it was set).

I = IFBRK(IDUM)

↑
returned as: 0 if break bit is not set
negative if break bit is set

Your program can then take an appropriate course of action.

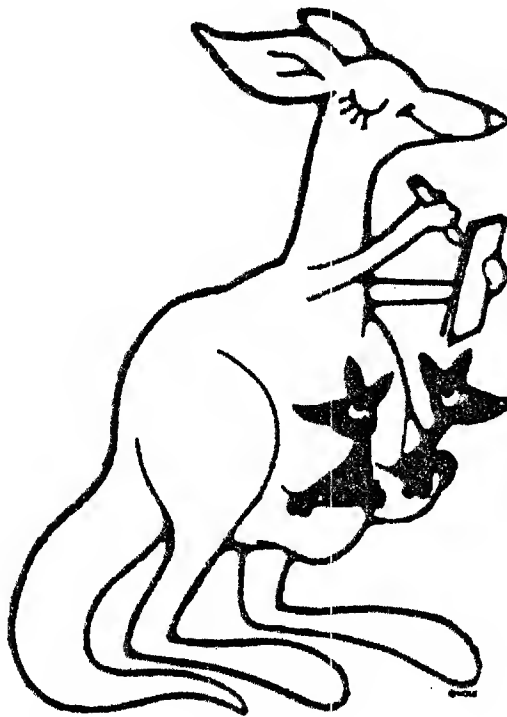
You can set a program's break bit with the system BR command.

S=xx COMMAND? BR,program

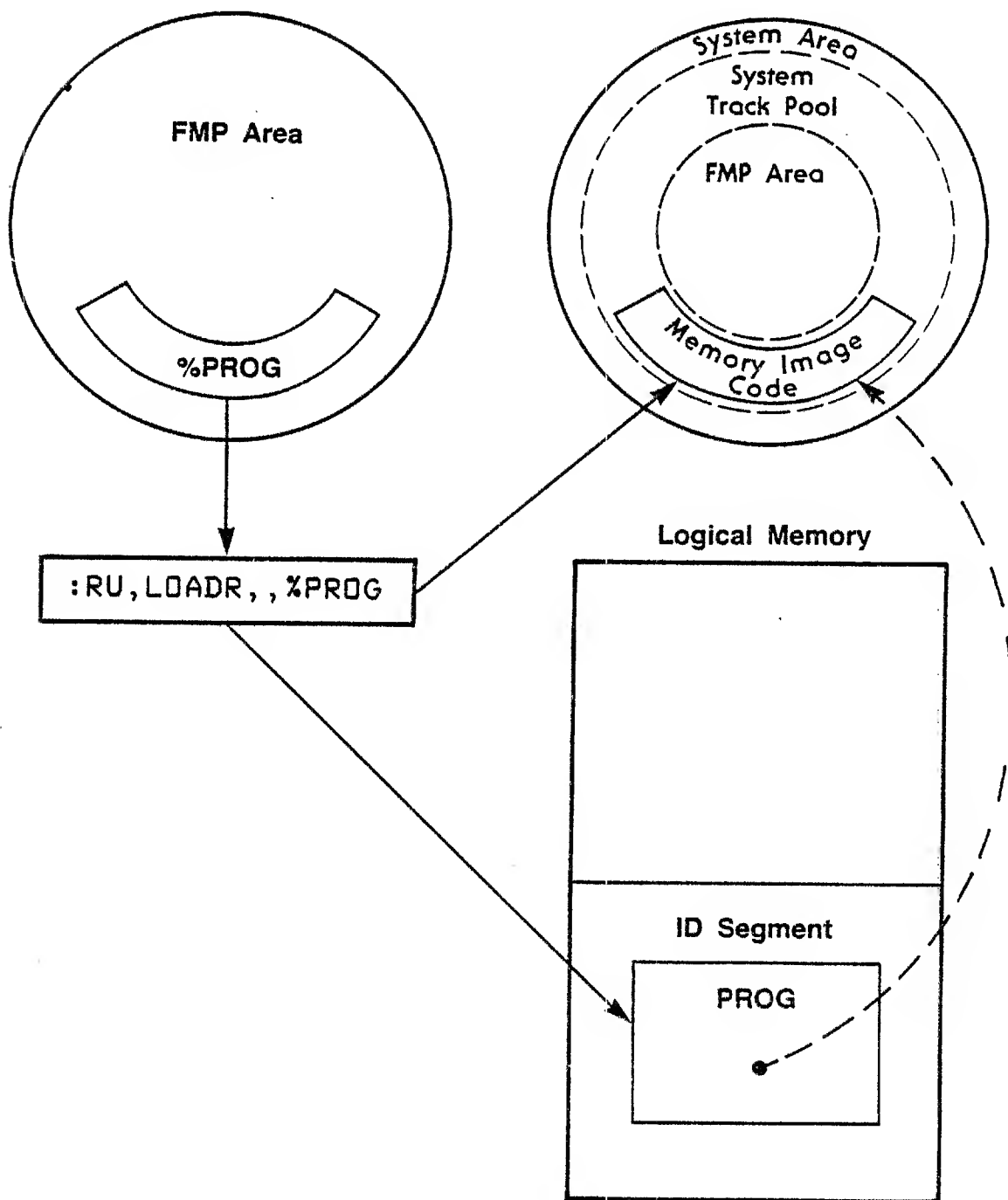
if "program" is not specified, the break bit in the current session program is set.

9

TYPE 6 FILES



LOADR CREATES A TEMPORARY DISC RESIDENT PROGRAM



SAVING PROGRAMS

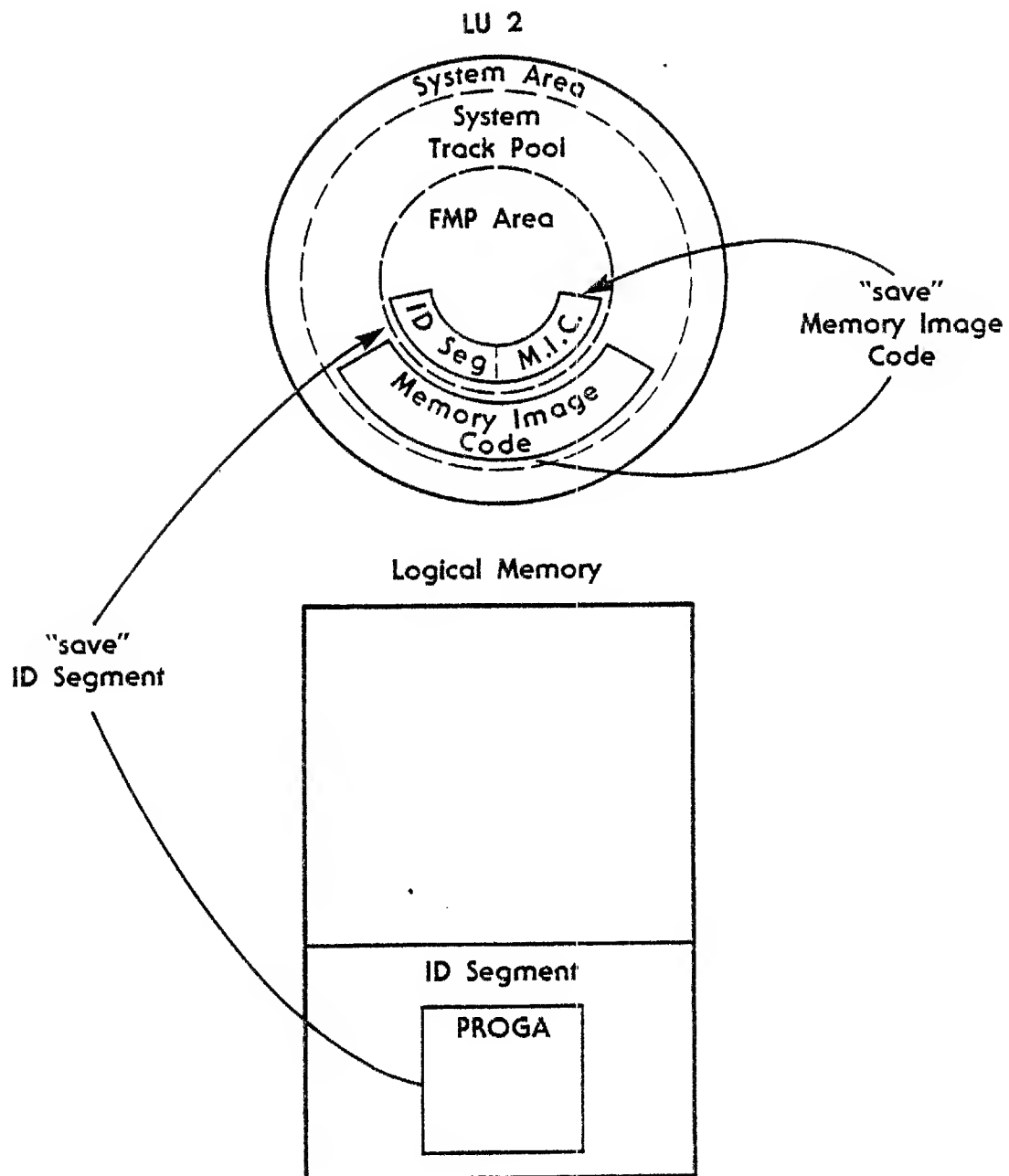
You can use the FMGR SP command to "save" a program as a type 6 file.

```
:SP, namr [ ,PR
           ,GR [ ,capability]
```

The type 6 file created by the SP command consists of a copy of the program's ID segment and Memory Image Code appended together.

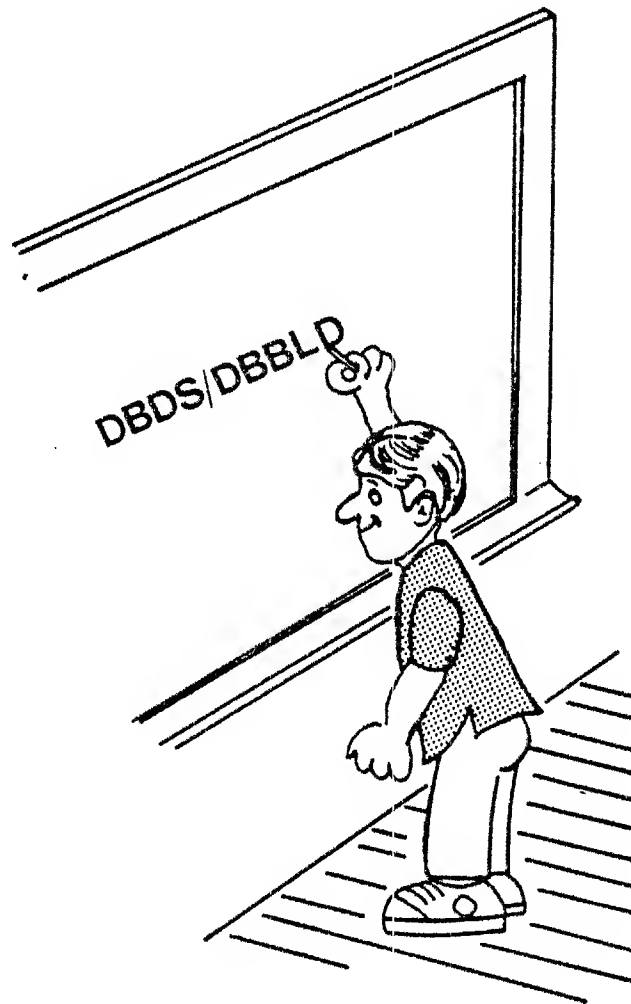
- * Type 6 files are not affected by the log-off process or the boot-up process.

: SP, PROGA



APPENDIX A

LAB EXERCISES



LAB 1 - AN INTRODUCTION TO RTE

1. Your instructor should supply you with the following information

_____ the Disc Loader ROM Number
_____ the Octal Select Code of the System Disc
_____ the Surface Number on the System Disc
which contains the RTE system

for the system you will use for your lab exercises.

Use this information to boot up the system.

LAB 2 - USING YOUR RTE SYSTEM

1. Log on to the system using the account given to you by your instructor.

What are the Session LU's which you can access?
What are their corresponding System LU's?

2. Retrieve any messages waiting for you, without purging them. Retrieve the messages again, this time purging the messages so you will not see them again. If you then enter a ME command, what will happen?

3. File &LAB23 contains a FORTRAN source program with a number of errors. You should -

- * create a copy of the source file for your own use, giving it a unique file name.
- * edit the source file, correcting any errors and giving the program a new name.
- * compile, load and run the program.

4. File &LAB24 also contains a FORTRAN source program with a number of errors. Repeat the steps of problem 3 with this source program.

LAB 3 - RTE ORGANIZATION

1. For every Session LU in your SST, determine its -

- a. corresponding System LU
- b. Eqt and subchannel number
- c. select code
- d. driver

2. File &LAB32 contains a FORTRAN source program which outputs several lines to the line printer. Make a copy of the source file for your own use and give the program a unique name. Then compile and load the program.

Before you run the program, turn the line printer off-line. Run the program and see what happens. Turn the printer back on-line and get your program's output.

Try the same exercise again but this time run the program with the System (RTE) run command.

(How do you enter a System command from FMGR?)

3. You can submit a System command with the FMGR SY prefix but you can't submit a Breakmode command this way. Try using the SY prefix with the Breakmode commands and see what happens.

(See the Terminal User's Manual description of the SY command.)

4. What LU and track(s) contain the Memory Image Code for the program you used for problem 2?

5. Use two different methods to find the priority of your copy of FMGR.

6. a) Enter the following command at your terminal -

:DL

What three commands can you enter to stop the output and get your FMGR prompt back?

- b) File &LAB36 contains a FORTRAN program which goes into an infinite loop, printing messages at your terminal. Make a copy of the source file for your use (with a new program name, of course) and then compile, load and run the program. Use the three commands you selected above and note what happens.

LAB 4 - FILE MANAGEMENT SYSTEM

1. Allocate (mount) a private cartridge from the Spare Cartridge Pool.

What is the LU of the cartridge allocated?

2. What are the names of all the type 6 files on LU 2?
The names of all the source files (first character of the file name an "&") on the system "lab exercises" cartridge?
3. The FORTRAN program in source file &LAB43 contains READ statements directed to LU 5 and WRITE statements directed to LU 6.
 - * Create a copy of the source file for your use. Give the program a new name but DO NOT change the I/O statements.
 - * Compile and load your program.
 - * Run the program but cause the input and output to be directed to your terminal.
4. What are two ways (two FMGR commands) to tell if there is a file called &LAB31 on the system "lab exercises" cartridge?
5. Use the FMGR ST command to store a file containing relocatable code onto a minicartridge (or magnetic tape). Then use the ST command to input the relocatable code from the tape into a new disc file.

Can you repeat this exercise using the FMGR DU command instead of the ST command?

6. Store 4 files on a minicartridge or a magnetic tape.

- * use the FMGR CN command to rewind the tape
- * use the CN command to position the tape (forward space) to the third file
- * use the FMGR DU command to list the third file at your terminal
- * use the CN command to backward space the tape to the second file
- * use the DU command to list the 2nd file at your terminal

Could you use the FMGR LI command instead of the DU command?

7. Use the FMGR LI command to list a source file. List the file again but specify the "B" option. Compare the two listings.

LAB 5 PROGRAM DEVELOPMENT

1. File &LAB51 contains a FORTRAN source program with the following control statement:

FTN4,L

Without editing the source file (or a copy), compile the program with

- a mixed listing
- a "Q" listing
- an "L" listing with a symbol table

What is similar about the mixed and the "Q" listings?

2. Make a copy of source file &LAB52, giving the program a new name. Compile the program and load it as a "real-time" program. Use LOADR to verify that the program really is a "real-time" program. Finally, run the program.

3. Write a FORTRAN program whose source code is contained in two files. One file should contain a main program which calls a subprogram; the other file should contain the source code for the subprogram.

After you compile the two source files, load the program using LOADR

- interactively
- from a command file

Could you use CLOAD to compile and load this program?

4. Files %GLIB and %XGLIB contain the relocatable code of two library files.

File &LAB54 contains a FORTRAN source program which uses some of the routines in these two libraries. Make a copy of the source program, giving the program a new name. Compile and load the program, searching the two library files for any needed routines (search %XGLIB before %GLIB). If you want to run the program you need to use a 2648 terminal.

Try loading the program using the LOADR LIBRARY command (see the Terminal User's Manual for a description of the LIBRARY command).

How could you determine the names of all the routines in each of these two library files?

5. Write two FORTRAN programs which will communicate with each other via a word in System Common.

Program 1 should enter a loop, requesting an integer value from the operator and storing this value in System Common. When the value 555 is entered, the program should store the value in System Common, jump out of the loop and terminate.

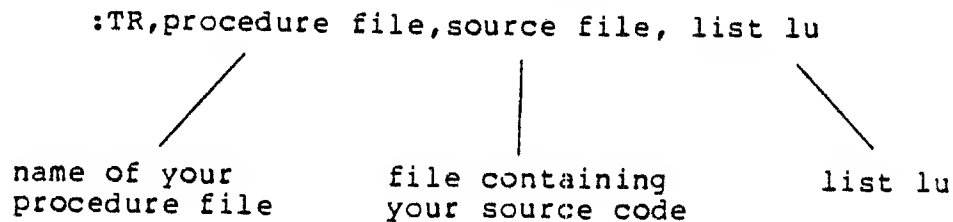
Program 2 should also enter a loop, printing the value it finds in System Common on a terminal. When the value 555 is found, the program should print this value, jump out of the loop and terminate.

Some suggestions:

- be careful when using System Common if someone else is also doing this exercise.
- run the two programs from separate terminals so they don't interfere with each other when doing I/O.

LAB 6 - PROCEDURE FILES

1. Log on as user LOW.CAPABILITY and see what this user can do.
List the user's HELLO file ("LOWCP::2").
2. What FMGR command can you use to determine your account's command capability level?
3. What are 4 ways to determine your Session Identifier (the System LU of your terminal)?
4. Write a procedure file to compile, load and run a program.
The procedure file is to be invoked with



- version 1 - ignore the possibility of compiler or LOADR errors.
- 'OF' the program after it is run to avoid getting duplicate program errors on subsequent loads.
 - use a file called %TEMP to hold the relocatable code. Will your procedure file work equally well if the relocatable file is called TEMP? See the RTE FORTRAN IV Reference Manual, appendix on compiler operations.
- version 2 - if a compiler error(s) occurs, print a message and transfer back to the operator.
- version 3 - if a LOADR error(s) occurs, print a message and transfer back to the operator.
- version 4 - require that the first character of the source file name be an "&". If not, print a message and transfer back to the operator.
- version 5 - make use of the "&" as the first character in the source file name by specifying "-" as the relocatable file in the :RU,FTN4.... command.

5. Write a procedure file which will store 4 disc files on a mini-cartridge or a magnetic tape. Store the procedure file on the tape as the first file, then invoke the procedure file to store the 4 files on the tape. Your tape should then look like this -

SOT

EOT

procedure file	file 1	file 2	file 3	file 4
-------------------	--------	--------	--------	--------

Transfer the 4 files back from the tape into 4 new disc files by

- storing the procedure file from the tape to a new disc file.
- editing the procedure file just input from tape so it will store 4 files from tape into 4 new disc files.
- invoking the new procedure file to read the 4 files from the tape.

The EDITR "X" command will be of use in editing the procedure file.

6. Write a procedure file which will accept the name of an existing file and then create a copy of that file for you with a new name. For example,

:TR,procedure file,file name

if you specified FILED, the procedure file will create a file called "FILExx", where "xx" is your Session Identifier, and store the contents of FILED into FILExx.

Suggestions:

- assume the file name to be at least 4 characters
- use the FMGR ST command to create the new file and transfer the data from the original file to the new file.

LAE 7 - USING RTE'S SERVICES PROGRAMMATICALLY

1. Files &LB711 and &LB712 each contain a FORTRAN source program. Each program prompts for an LU and then outputs a message identifying itself on that device.

Make a copy of each source file, giving the programs new names. Then compile and load the two programs.

For this exercise, you will need to use a terminal in addition to your own. With the second terminal displaying a FMGR prompt,

- Set the priority of one program to 25.
Set the priority of the other to 30.
- Run the priority 30 program, directing the message to the second terminal.
Run the priority 25 program, also directing the message to the second terminal.
- Hit the return key on the second terminal.

In what order do the messages appear? Repeat this exercise but use priorities of 85 and 90.

2. Using EXEC READS and WRITES, write a program which prompts the operator to enter up to 40 characters and then prints those characters (and only those characters) on the line printer. Have your program repeat this process until the operator enters a null string (just a carriage return) in response to the prompt issued by your program.

If you like, you can use a FORTRAN WRITE for the prompt.

3. Write a program which inputs an integer value and then sets the time-out of your terminal to that value.

Be careful about the value you enter since time-outs are in units of 10's of milliseconds!

If you like, you can use FORTRAN READS and WRITES for the program's I/O; use an EXEC Call to set the terminal's time-out.

4. Write a program which prompts the operator for an LU number and then prints a message stating whether that LU is

- a terminal
- a magnetic tape drive
- a printer
- none of the above

Again, you can use FORTRAN statements for the I/O; use an EXEC Call to determine the device information.

What happens if you enter an LU not in your SST?

Modify your program to print a message if you specify an LU not in your SST.

LAB 8 - INTERACTING WITH YOUR PROGRAM

1. Write a program that prints a message on a device to be specified when the program is run. For example,

:RU,program,6

print the message on LU 6
(assume non-disc devices)

2. Write a program that tells you how many characters are in your first name. Design your program so that it can be run in two ways -

:RU,program,name

(The program determines the numbers of characters in the specified name.)

-or-

:RU,program

(The program should prompt the user to enter a name.)

3. File &LAB83 contains a FORTRAN source program which accepts a string of 4 to 6 ASCII characters and renames the string for your Session. For example,

:RU,program,FILEDD

The program uses PRTN to return the string "FILExx", where "xx" is the Session Identifier of the user who ran the program.

If the passed string has less than 4 or more 6 characters, the program will print a message and pass back a 0 in the first return parameter.

Make a copy of the source file (giving the program a new name), then compile and load the program.

Look at the procedure file you wrote for problem 6 of the procedure file lab (a procedure file to accept the name of a file and create a copy of the file with a new name, determined by your Session Identifier).

Modify the procedure file so that it uses this program to determine the new file name. Your procedure file can then use this new name in the FMGR ST command to create the new file and transfer the data from the original file to the new file.

4. Write a program which goes into a loop, writing lines on your terminal until you "break it." Your program should then suspend itself. When you reschedule the program, include a parameter which will specify one of two options:

- print 1 line on the line printer and then stop
- stop without doing any more printing.

While your program is suspended, use WHZAT to determine its state.

LAB 9 - TYPE 6 FILES

1. Write a FORTRAN program which contains an infinite loop, printing a message at your terminal. After compiling and loading your program,
 - run the program
 - terminate the program with the system 'OF' command
 - save the program as a type 6 file
 - release the program's ID Segment and disc tracks containing its Memory Image Code
- .- run the program
 - terminate the program with the system 'OF' command.

LAB 10 - USING FILES PROGRAMMATICALLY,
FMP CALLS

* PART A *

1. File TLB101 contains an ASCII text file whose maximum record length is 10 words.

Write a program to list the contents of the file at your terminal.

Some considerations -

Remember that other people may be using the file at the same time you are.

You might write your program so you can "break it."

2. Write a program that will create a type 2 file with

length - 5 blocks
record length - 1 word

Your program should store the value 1 into the first record, the value 2 into the second record, the value 3 into the third record and so on.

After your program terminates, use the FMGR LI command to verify the operation of your program.

3. File &LB103 contains a FORTRAN source program designed to modify the file you created in problem 2. The program will
 - open the file,
 - store the value 7777B into the first record and then
 - terminate.

Make a copy of the source file, giving the program a new name. Compile, load and run the program. Then use the FMGR LI command to check the program's operation.

4. File VLB104 is a type 1 disc file containing 640 real values.

Write a program to input the values and then calculate and print the average of the values.

